VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA

EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Návrh a implementace metodiky dle SCRUM v softwarové firmě

Design and Implementation of Methodology According SCRUM in Software Company

Student: Bc. Petr Martinásek

Vedoucí diplomové práce: Ing. Jan Ministr, Ph.D.

Ostrava 2013

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání diplomové práce

Student: **Bc. Petr Martinásek**

Studijní program: N6209 Systémové inženýrství a informatika

Studijní obor: 1802T001 Aplikovaná informatika

Téma: Návrh a implementace metodiky dle SCRUM v softwarové firmě
Design and Implementation of Methodology According to SCRUM in Software Company

Zásady pro vypracování:

1. Úvod
2. Metodická východiska a nástroje objektově orientovaného vývoje software
3. Analýza stávajícího stavu použití metodik ve firmě
4. Návrh a implementace metodiky podle SCRUM
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků diplomové práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. Brno: Computer Press, 2007. 567 s. ISBN 978-80-251-1503-9.
BRITTON, Carol a Jill DOAKE. *A student guide to object-oriented development*. Amsterdam: Elsevier Butterworth-Heinemann, 2005. 416 s. ISBN 978-075-0661-232.
SCHWABER, Ken a Mike BEEDLE. *Agile software development with Scrum*. Upper Saddle River: Prentice Hall, 2001. 158 s. ISBN 01-306-7634-9.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Ministr, Ph.D.**

Datum zadání: 23.11.2012
Datum odevzdání: 26.04.2013

Ing. Petr Rozehnal, Ph.D.
vedoucí katedry

prof. Dr. Ing. Dana Dluhošová
děkanka fakulty

Prohlašuji, že jsem celou práci vypracoval samostatně. Příloha A daná mi k dispozici, jsem samostatně doplnil.

V Ostravě dne 25.4.2013 .....

Poděkování

Rád bych poděkoval vedoucímu diplomové práce Ing. Janu Ministrovi, Ph.D. za vstřícný přístup, odborné vedení a množství rad.

Dále bych chtěl poděkovat Mgr. Lence Mališové ze společnosti D3Soft s.r.o. a Ing. Tomášovi Frydrychovi ze společnosti Poski.com za poskytnutí materiálů a informací nezbytné pro vypracování této práce.

Lingvistická poznámka

Výčet všech použitých zkratek, bude uveden na konci této práce. Některé zkratky pro pochopení textu budou vysvětleny i v průběhu práce, a to při jejich prvním požití, kdy nejdřív bude uvedena zkratka a za ní v závorkách vysvětlení zkratky. V případě zkratky z anglického názvu, bude v závorce uveden i český překlad, oddělený za anglickým názvem pomlčkou. Většina termínu bude vysvětlena taktéž v průběhu práce, avšak diplomová práce má technický charakter, proto se v ní bude vyskytovat odborná a cizí terminologie, které se ve vědecké společnosti běžně užívají a jsou obecně známé (software, internet aj.). Pro některé anglické termíny (např. Scrum Master) a obzvlášť u metodik (Feature-Driven Development, Crystal aj.) nejsou často obecně uznávané české ekvivalenty a používají se jejich anglické formy (v literatuře jsou překládány různě a mohou mást čtenáře).

Při uvádění anglických názvů se však budeme snažit většinu přeložit dle anglicko-českého slovníku nebo uvedeme český ekvivalent, který se v literatuře běžně užívá a pojmenovává stejný prvek, i když nejde o jeho doslovný překlad (handle). V těchto případech nebudeme uvádět český překlad v závorkách za jeho anglickým ekvivalentem.

Také česká terminologie v softwarovém inženýrství není ustálená a jednotná. Proto vymezíme použité základní termíny tak, aby při jejich použití v textu byl jasný jejich význam. Některé termíny mají mnoho ekvivalentů, které mohou mít za různých okolností trochu odlišný význam, my je v této diplomové práci budeme chápat jako synonyma.

Vývojový proces, vývoj softwaru, softwarový proces, výrobní proces, projekt, výroba softwaru, tvorba softwaru – časově ohraničené lidské úsilí, na jejímž konci je softwarový produkt nebo služba.

Zhotovitel, vývojáři, vývojový tým, tým – skupina lidí, kteří na zakázku vytvářejí určitý software.

Zadavatel projektu, zákazník, objednavatel, klient – subjekt, který zadal poptávku po novém softwaru.

Software, systém, program, aplikace, produkt – výsledný produkt procesu vývoje softwaru, který lze zobrazit přes internetový prohlížeč a využívá pro komunikaci internet.

Obsah

1 Úvod

Díky dnešní obrovské konkurenci jsou všechny firmy napříč odvětvími nuceny neustále zefektivňovat své výrobní procesy. Snaží se dodávat kvalitnější produkty než konkurence, v co nejkratším čase a s co nejmenšími náklady. Aby toho dosáhly, snaží se vyrábět podle nějaké nadefinované šablony, kterou mohou dále optimalizovat a být tak krok před konkurencí. Stejně je tomu tak i při vývoji softwaru. Jak vyvíjet software s vysokou kvalitou, s nízkými náklady a s minimálním časem se zabývá obor softwarové inženýrství, do kterého spadá také tato diplomová práce. Bohužel, většina firem pohlíží na tuto disciplínu stále jako na problematiku, kterou by se měla zabývat až postupem času a pouze pokud na ni zbude čas a peníze. Opak je však pravdou. Když bude firma vyvíjet software rychleji, stihne uspokojit více klientů za stejné náklady. Zefektivnit vývoj softwaru se snaží lidé pomocí různých metodik, metod, nástrojů, technik, norem a standardů. My se v této práci budeme zabývat především metodikami, konkrétně se zástupcem z metodik agilních, což je skupina metodik založená na iterativním a inkrementálním vývoji, s důrazem na rychlost a minimalizaci zdrojů. Agilní metodiky už dávno nejsou horkou novinkou. Jsou zde již několik let a používají je největší světové softwarové firmy, mezi které se chce zařadit také softwarová firma Poski.com. Tato firma si vybrala agilní metodiku Scrum, kterou chce přizpůsobit a zavést do svého vývojového procesu.

Cílem této práce bude navrhnout a implementovat metodiku dle Scrum do této softwarové firmy. Posloupnost kroků vedoucí k tomuto cíli bude dokládat popis jednotlivých kapitol. Celá práce bude rozdělena do tří částí.

První část bude obsahovat teorii o dané problematice, kde v první kapitole bude uvedení do problematiky softwarového inženýrství. V dalších kapitolách si rozebereme životní cyklus softwarového projektu, definujeme si faktory ovlivňující vývoj softwaru, uvedeme si normy a standardy. Následovat bude kapitola zabývající se obecně metodikami. To znamená ujasnění pojmů kolem metodik, vývoj metodik, rozdělení metodik, popis rigorózních a agilních metodik. Lehce nakousneme problematiku výběru metodik a stav metodik u nás i ve světě. Teoretická část bude dále obsahovat samostatnou kapitolu o metodice Scrum, její celý popis, charakteristiku a strukturu. Závěr teoretické části bude patřit nástrojům objektově orientovaného vývoje softwaru a modelovacímu jazyku Unified Modeling Language.

Abychom mohli navrhnout a implementovat nějakou metodiku do firmy, je nutné ji nejdříve zmapovat. Ve druhé části tedy provedeme analýzu stávajícího stavu použití metodik ve firmě, která se bude skládat z popisu firmy Poski.com, tzn., uvedeme její historii, činnost

a organizační strukturu. Popíšeme neúspěšný pokus nasazení metodiky Scrum před dvěma lety a zanalyzujeme aktuální vývojový proces z hlediska agilních principů a principů dle metodiky Scrum. Následovat bude SWOT analýza pro pochopení globálních faktorů. Na závěr analýzy společnosti Poski.com identifikujeme problémy, které budeme řešit v třetí části. Protože Scrum je pouze procesní rámec a neodpovídá na některé dílčí problémy, při návrhu se pokusíme nalézt odpovědi v jiné, větší firmě – D3Soft s.r.o., která již vyvíjí agilním způsobem a má jisté zkušenosti s problematikou vývojového procesu. Proto druhá část bude obsahovat také analýzu společnosti D3Soft, se zaměřením na nalezení řešení identifikovaných problémů v Poski.com a zároveň porovnání určitých aspektů vývoje SW.

Třetí část bude obsahovat návrh řešení identifikovaných problémů, pomocí metodiky Scrum, agilních principů a nejlepších praktik z firmy D3Soft. Na závěr třetí části se zaměříme na komplexní návrh metodiky a popíšeme implementaci metodiky ve formě kroků zavedení do firmy Poski.com.

Poslední kapitolou je závěr práce, který shrnuje dosažené výsledky a hodnotí přínos práce. A právě co možná největší přínos práce bude náš druhotný cíl. Kromě splnění vytyčeného primárního cíle (navrhnout a implementovat metodiku dle Scrum), bude naším druhým cílem snaha o reálné využití konečného návrhu v praxi. Ne vždy totiž teoretický/akademický pohled koresponduje s realitou. Práce sice bude pohlížet na problematiku z akademického hlediska, směřovat bude ale především k jejímu praktickému využití. V diplomové práci proto nebudeme navrhovat věci, které se v dané firmě nedají realizovat. Ověření tohoto sekundárního cíle bude problematické, protože prohlásit, že navržená metodika je vhodná do praxe a je efektivní, lze až po několika projektech a měsících od zavedení. Druhý cíl tedy bude pouze spíš udávat směr této práce.

2 Metodická východiska a nástroje objektově orientovaného vývoje software

V této kapitole budou představena metodická východiska a nástroje orientovaného vývoje softwaru. Nejdříve popíšeme obor softwarové inženýrství, do kterého tato diplomová práce spadá, jeho historii a krize. Rozebereme a popíšeme životní cyklus softwarového projektu a faktory, které jej ovlivňují. V následující kapitole se budeme zabývat standardy a normami pro vývoj softwaru. V další kapitole se zaměříme na metodiky, které budou obsahovat jejich rozdělení, vývoj metodik, popis jednotlivých metodik, jak vybrat správnou metodiku a stav metodik u nás i ve světě. Metodice Scrum bude věnována samostatná kapitola. Poté popíšeme nástroje objektově orientovaného vývoje softwaru a na závěr charakterizujeme normu pro vizualizaci modelů – UML (Unified Modeling Language – unifikovaný modelovací jazyk).

2.1 Softwarové inženýrství

Problematikou vývoje softwaru se zabývá obor softwarové inženýrství. Po prostudování odborné literatury, můžeme najít několik definicí softwarového inženýrství. My si zde uvedeme tři verze:

Definice IEEE 1993: „Softwarové inženýrství je systematický, disciplinovaný a kvalifikovaný přístup k vývoji, tvorbě a údržbě softwaru“ (Mannová a Vosátka, 2005, s. 13).

Druhá definice zní takto: „Softwarové inženýrství je zavedení a používání řádných inženýrských principů tak, abychom dosáhli ekonomické tvorby softwaru, který je spolehlivý a pracuje na dostupných výpočetních prostředcích“ (Kadlec, 2004, s. 22).

Definice IEEE 610.12: „Softwarové inženýrství je aplikace systematického, disciplinovaného, kvantifikovatelného přístupu k vývoji, provozu a údržbě softwaru, tj. aplikace inženýrství na software. Také je to studium přístupů dle výše uvedeného“ Richta (nedatováno, s. 2).

Základní kritéria této disciplíny jsou kvalita, čas a zdroje. Jakékoliv z těchto kritérií se však může v průběhu vývoje softwaru změnit. Ať je to personální změna ve vývojovém týmu, změna termínu dodání nebo rozpočet vymezený pro daný projekt. Stejně tak se mohou objevit nové požadavky na daný systém nebo se mohou změnit již staré specifikované na začátku projektu. S těmito skutečnostmi však musí vývojový tým počítat a být na tuto možnost připraven (Mannová a Vosátka, 2005).

Softwarové inženýrství je k poměru ostatním vědám, obor mladý. (Až v 1997 bylo softwarové inženýrství uznáno jako samostatný obor s certifikátem v USA (Kadlec, 2004). Avšak jeho rozvoj je velmi dynamický a jeho význam velmi roste. Většina manažerů a firem si tuto skutečnost více a více uvědomuje (Mannová a Vosátka, 2005).

Softwarové inženýrství vzniklo v roce 1968 jako reakce na chaotický vývoj softwaru. Doba před vznikem oboru je známá jako tzv. softwarová krize.

Příčiny softwarové krize podle Kadlece (2004).

- Špatná komunikace, jak ve vývojovém týmu, tak mezi vývojáři a klientem.
- Nesprávný přístup. Programátor byl upřednostňován před klientem.
- Špatné plánování celého projektu.
- Nesprávné odhady doby trvání vývoje projektu, ceny projektu, rozsahu projektu.
- Nízká produktivita práce, špatné stanovení priorit práce.
- Neznalost základních pravidel, která vznikla ze zkušeností s vývojem softwaru.
- Podcenění hrozeb a rizik.
- Nezvládnuté technologie. Zavádění nových technologií, které však ještě nebyly zvládnuty.

Důsledky těchto příčin byly: nízká kvalita programů, neúnosné prodlužování a prodražování projektů, nemožnost údržby a inovace, špatná efektivita práce programátorů i celého týmu, neefektivita vývoje, nejistota výsledku a řada dalších. (Vondrák, 2002), (Mannová a Vosátka, 2005), (Martinásek, 2011).

Na tyto problémy se snaží odpovědět různé metodiky. V 80. letech minulého století to byly metodiky strukturované a následně objektově orientované. V 90. letech se počet metodik rozrostl o největší počet, vzniklo několik nejvíce známých metodik, včetně metodik agilních. Těmito a dalšími metodikami se budeme podrobněji zabývat v kapitole 2.6 Metodiky.

2.2 Životní cyklus softwarového projektu

V této kapitole se budeme zabývat popisem a životního cyklu softwarového díla, který je někdy také označován jako softwarový proces, z jakých fází se skládá a co je předmětem těchto fází.

„Softwarový proces je po částech uspořádaná množina kroků směřujících k vytvoření nebo úpravě softwarového díla“ (Vondrák, 2002, s. 8).

Z výše uvedené definice se softwarový proces skládá z množiny kroků. Pro řízení těchto kroků je efektivní je sloučit do podmnožin neboli fází, tyto fáze jsou v literatuře definované různě, i když často se liší jen v nepatrných odchylkách. Některé metodiky tyto fáze slučují, některé metodiky určitou fází nedefinují vůbec. Naopak například metodika RUP (Rational Unified Process) životní cyklus projektu rozšiřují o některé další fáze. Fáze se mohou překrývat nebo navazovat, v některých případech dokonce začínat ihned na začátku projektu a končit na samém závěru projektu. My se k řadě autorů, kteří definují fáze vývoje softwaru, přidáme a pokusíme se nadefinovat vlastní fáze. Hlavní inspirací nám bude již zmíněná metodika RUP (IBM, 2007), (Aldolf, 2008b). Dalšími zdroji nám budou pro inspiraci autoři: Mannová (2005), Hradecká (2008), Kendall E. a Kendall J. (2010) a Kent Beck (2002).

Z následujícího rozdělení je vidět snaha o co největší fragmentaci. Fáze jsou uvedeny v její teoretické posloupnosti a návaznosti, v praxi to ale nemusí být pravidlo.

- Byznys modelování – modelování podnikových scénářů, převáženě pomocí UML. U nezávislých systémů na určitý podnik se tato fáze vypouští.
- Specifikace požadavků – zjištění, popis, formalizace a zpracování požadavků na systém. Požadavky zahrnují především funkčnost a design systému, ale také návaznost a integrace s jinými systémy.
- Analýza – analyzování systémových potřeb nezbytných pro tvorbu systému, stanovení globální strategie, identifikace problému, možností a cílu.
- Návrh – nadefinování architektur systému včetně jeho členění na komponenty (návrh databáze).
- Implementace – naimplementování navržených modelu do spustitelné podoby, dokumentace softwaru.
- Testování – vytvoření a provedení souboru testů pro verifikaci funkčnosti komponent systému a jejich korektní integrace.
- Nasazení – předání a poskytnutí funkčního systému koncovým uživatelům.
- Udržování systému – zajišťování bezproblémového provozu systému.
- Rozvoj systému – upravování, rozšiřování, doplnění a vylepšení softwarového produktu.
- Stažení systému – stáhnutí systému z jeho užívání a ukončení projektu.
- Vyhodnocení projektu – zhodnocení dosažených cílu, získání a dokumentace nově nabytých poznatků a zkušeností vedoucí k optimalizaci a dalších projektů.

Další podpůrné činnosti během vývoje, které však nejsou brány jako fáze vývoje softwaru, jsou:

- správa konfigurací,
- řízení projektu,
- a příprava prostředí.

2.3 Faktory vývoje softwaru

Pokud chce nějaká společnost efektivně vyvíjet kvalitní software, musí si plně uvědomovat všechny faktory, které jej mohou ovlivňovat. Jednu z nejvýznamnějších rolí v procesu vývoje softwaru hraje lidský faktor. Dalšími faktory, které by měl mít na paměti každý softwarový inženýr, manažer nebo vedoucí projektu podle Buchalcevové (2005a) a Vondráka (2002) jsou:

- dostupnost kvalifikovaných specialistů, expertů a odborníků,
- technické aspekty – počítačová infrastruktura a softwarové vybavení,
- stabilita technologie pro implementaci (nové technologie jsou méně stabilní),
- stabilita a schopnosti nástrojů,
- efektivnost používaných metod,
- nová funkcionalita a její vztah k existující funkcionalitě,
- metodika a její flexibilita,
- čas a zdroje (ekonomické možnosti),
- management – efektivní řízení a využití všech dostupných zdrojů s cílem vytvořit produkt požadované kvality,
- klienti – jeden z nejvíce podceňovaných faktorů, který si právě klienti neuvědomují, naštěstí agilní metodiky se snaží tento problém eliminovat,
- konkurence,
- a další proměnné.

Efektivním využitím uvedených faktorů se zabývají různé metodiky, metody, nástroje, techniky, normy a standardy, kterými se budeme zabývat v následujících kapitolách.

2.4 Normy a standardy

Pro vývoj softwaru, jako dnes ve většině lidských činností, existují také zde normy a standardy, které pomáhají svými zásady k dodržení kvality a normovaných postupů.

2.4.1 Standard pro řízení a správu IT služeb

Existuje několik souborů pravidel pro efektivní vývoj softwaru. Jeden z nejznámějších takovýchto souborů konceptů a postupů, které umožňují lépe plánovat a zkvalitňovat využití informačních technologií je například ITIL (Information Technology Infrastructure Library). „ITIL je mezinárodně uznávaný a rozšířený de-facto standard pro řízení a správu IT služeb“ (Management Mania, 2013a, s. 1). ITIL tedy není normou nebo standardem, obsahuje pouze doporučení pro zvládnutí řízení informačních technologií v společnosti. Z ITIL však vychází norma ISO 20000. Standard ITIL byl vyvinut OGC v 80. letech 20. století a aktuálně je dostupný ve verzi ITIL V3.

ITIL vychází z nejlepších zkušeností vývojářů. Zaměřuje se na neustálé měření a zlepšování kvality dodávaných služeb IT, jak ze strany dodavatelů, tak ze strany zákazníka. Cílem ITIL je zajištění optimální služby zákazníkům s největší možnou kvalitou (Cartlidge at al., 2007), jak toho docílit dokládá obrázek 2.1.

ITIL podle Kuchaře (nedatováno) popisuje: co se má provést, kdo a s čím se pracuje a kolik toho lze poskytnout. ITIL naproti tomu se nezabývá, jak mají být procesy přesně implementovány a prováděny.

Obr. 2.1 Service Profit Chain model (Kuchař, nedatováno)

Součástí ITIL je také SLA (Service Level Agreement) smluvní dokument mezi zákazníkem a poskytovatelem, který definuje předmět, neboli popis služby, rozsah, úroveň a kvalitu služby, cenu a podmínky poskytování služeb. (Buchalcevová, 2005a). Překládá se jako dohoda o úrovni poskytovaných služeb.

Definováním podmínky služeb můžeme rozumět: způsob řešení podpory zákazníků, komunikační kanály mezi zákazníkem a poskytovatelem, způsob řešení výjimečných nebo havarijních stavů, garantovaná rychlost reakce a odstranění poruchy, garantovaná časová dostupnost služby, stanovení odpovědností za škody, řešení duševních a autorských práv a další (Management Mania, b). Zpoplatnění pak bývá řešeno dvojím způsobem:

- měsíčním paušálem,
- zpoplatnění je závisle na objemu a kvalitě dodaných služeb v daném období (Voříšek, 2008).

Další relevantní veřejné rámce a normy pro poskytování rad a návodů o nejlepších praktikách pro poskytování IT služeb jsou ISO/IEC 20000: správa služeb IT a ISO/IEC 27001: správa bezpečnosti informací, COBIT (Control Objectives for Information and related Technology), PRINCE (Projects in Controlled Environments), PMBOK (Project Management Body of Knowledge), MOR (Management of Risk), eSCM–SP (eSourcing Capability Model for Service Providers), eTOM (Telecom Operations Map), Six Sigma (Cartlidge at al., 2007) a konečně CMMI, kterým se bude zabývat v další kapitole.

2.4.2 Capability Maturity Model Integration

Společnosti zabývající se vývojem softwaru se dají podle CMMI (Capability Maturity Model Integration - stupňovitého modelu vyspělosti) rozdělit do pěti úrovní. A to podle toho, jak kvalitně dokážou vyvíjet software. Model byl vytvořen v instituci SEI (Software Engineering Institute), která pracuje v rámci Carnegie Mellon University. Model vznikl za účelem definování cílů a doporučených pracovních postupů pro vývojové týmy, které by mělyi vést k efektivnímu vývoji produktu s odpovídajícím kvalitním výstupem (Vondrák, 2002), (Mannová a Vosátka, 2005). Následuje výčet úrovní vyspělosti.

1. Initial (Počáteční) – firma nemá definován softwarový proces vývoje SW a každý projekt je řešen různým způsobem.
2. Repeatable (Opakovatelná) – firma našla v jednotlivých projektech opakovatelné postupy a je schopna tyto postupy v dalších nových projektech opakovaně používat.
3. Defined (Definovaná) – nalezený softwarový proces je firmou definován a dokumentován. Firma se podle něj řídí.
4. Managed (Řízená) – na základě definovaného a dokumentovaného softwarového procesu je firma schopna proces řídit a monitorovat.

5. Optimized (Optimalizovaná) – na základě dlouhodobého monitorování softwarového procesu jsou získány zpětně vazby, které jsou využívány k jeho optimalizaci.

2.4.3 Normy a standardy pro životní cyklus softwaru

Existuje také několik mezinárodních standardů pro procesy v rámci životního cyklu. Jeden z nich je ISO/IEC 12207, publikován v roce 1995. Standart zahrnuje vývoj, dodávky, podporu a údržbu softwaru. Podpoře procesu životního cyklu a organizačním procesům. (ISO, 2008).

Druhým z mezinárodních standardů, který si zde uvedeme je ISO/IEC 15504 známý pod zkratkou SPICE. Tento standard hodnotí proces vývoje softwaru. Byl odvozen ze standardu procesu životního cyklu ISO 12207 a maturity models (modely vyspělosti) jako je model CMMI (popsán v předchozí kapitole), Bootstrap a Trillium. Standard obsahuje takzvané dimenze procesu, které jsou rozděleny do pěti kategorií procesu na dimenzi odběratel/dodavatel, inženýrství, podpora, řízení a organizace. Každá tato dimenze procesu je hodnocena pomocí devíti kriterií: výkonnost procesu, řízení procesu, dále definice, rozmístění, měření, kontrola, inovace a optimalizace procesu (ISO, 2004).

V roce 2002 Úřad pro veřejné informační systémy v Praze vydal standard ISVS 005/02.1. Standard definuje základní postupy procesu životního cyklu informačního systému pro veřejnou správu (Úřad pro veřejné informační systémy, 2002).

Existuje samozřejmě ještě spousta dalších norem zajišťující kvalitu, bezpečnost, ochrany zdraví při práci a jiné. Ty se však specializují jen na dílčí problematiku vývoje, nebo nejsou specializovány na vývoj softwaru, ale jsou využity v širším spektru lidské činnosti.

2.5 Metodiky

V této kapitole se budeme zabývat metodikami. Nejdříve si ujasníme pojmy, jako jsou metodika, metodologie, metoda, nástroj a technika. Po té přejdeme k vývoji metodik. Zmíníme zásadní mezníky a metodiky, které nejvíce ovlivnily vývoj metodik, jako jsou vodopádový model nebo spirálový model. Uvedeme dnes nejznámější metodiky a rozdělíme je do skupin, podle úhlu pohledu na proces vývoje, na metodiky rigorózní a metodiky agilní. Rigorózním metodikám bude patřit další kapitola, kde se bude primárně zabývat jejím nejznámějším zástupcem – Unified Process, respektive jeho komerční aplikací Rational Unified Process. V této kapitole se zmíníme také o takzvaných nejlepších praktikách, které většina společností, zabývající se vývojem softwaru, snaží dodržovat, i když se neřídí při vývoji metodikou Rational Unified Process. V další kapitole se budeme zaobírat druhou

skupinou metodik, metodikami agilními. Zmíníme zásady a praktiky agilních metodik, v čem se liší od metodik rigorózních a stručně si uvedeme ke každému zástupci agilních metodik několik základních informací. V předposlední kapitole z oblasti metodik jen okrajově nahlídneme do problematiky výběru metodik pro určitý projekt, tým, nebo společnost. Na závěr zanalyzujeme stav metodik u nás i ve světě, z pohledu jejich využívaní, respektive nevyužívání.

2.5.1 Pojmy a úvod do metodik

Na úvod je třeba si ujasnit základní pojmy v oblasti metodik, a metodiky samotné. Buchalcevová (2005a) metodiku vysvětluje takto: „Metodika (methodology) představuje v obecném smyslu souhrn metod a postupů pro realizaci určitého úkolu“ (Buchalcevová, 2005a, s. 13). Buchalcevová také vysvětluje co je cílem metodiky a kde se využívá: „Metodika je tvořena obecně uznávanými postupy a návody, které popisují činnosti při analýze, návrhu, vývoji, nasazování software stejně jako činnosti spojené s řízením projektu. Cílem metodiky je formalizovat postupy, definovat zodpovědnosti a pravidla komunikace“ (Buchalcevová, 2005a, s. 13).

„Metodika je doporučený souhrn etap, přístupů, zásad, postupů, pravidel, dokumentů, řízení, metod, technik a nástrojů pro tvůrce informačních systémů, který pokrývá celý životní cyklus informačních systémů. Metodika by se měla vztahovat na všechny prvky informačního systému, na pracovníky, data, software, hardware, organizační procedury, ekonomické otázky spojené s vývojem a provozem systému, doporučené dokumenty, způsoby řízení v jednotlivých fázích životního cyklu systému“ (Buchalcevová, 2005a, s. 13).

Nauka o metodikách se nazývá Metodologie, která rozebírá, zkoumá a definuje metodiky. (Kadlec, 2004)

Dalším důležitým termínem je Metoda. „Metoda je již určitým způsobem, jak provádět určitou věc systematickým způsobem. Je logicky uspořádána, tedy jednotlivé postupy jsou seřazeny v krocích“ (Kadlec, 2004). Arlow a Neustadt, (2007) metodu definují takto: „Metoda tvorby softwarového vybavení definuje při vývoji softwaru otázky, kdo, co, kdy a jak“.

Podmnožinou metodiky bývá technika, která udává, jak dojít k danému výsledku. Technika se skládá z přesně definovaných konkrétních kroků, činností a postupů jak používat určité nástroje. Technika jasně definuje závěry, je však více omezená ve svojí použitelnosti než metoda. Mezi techniky patří například transakční analýza nebo normalizace datového modelu. (Hradecká, 2008)

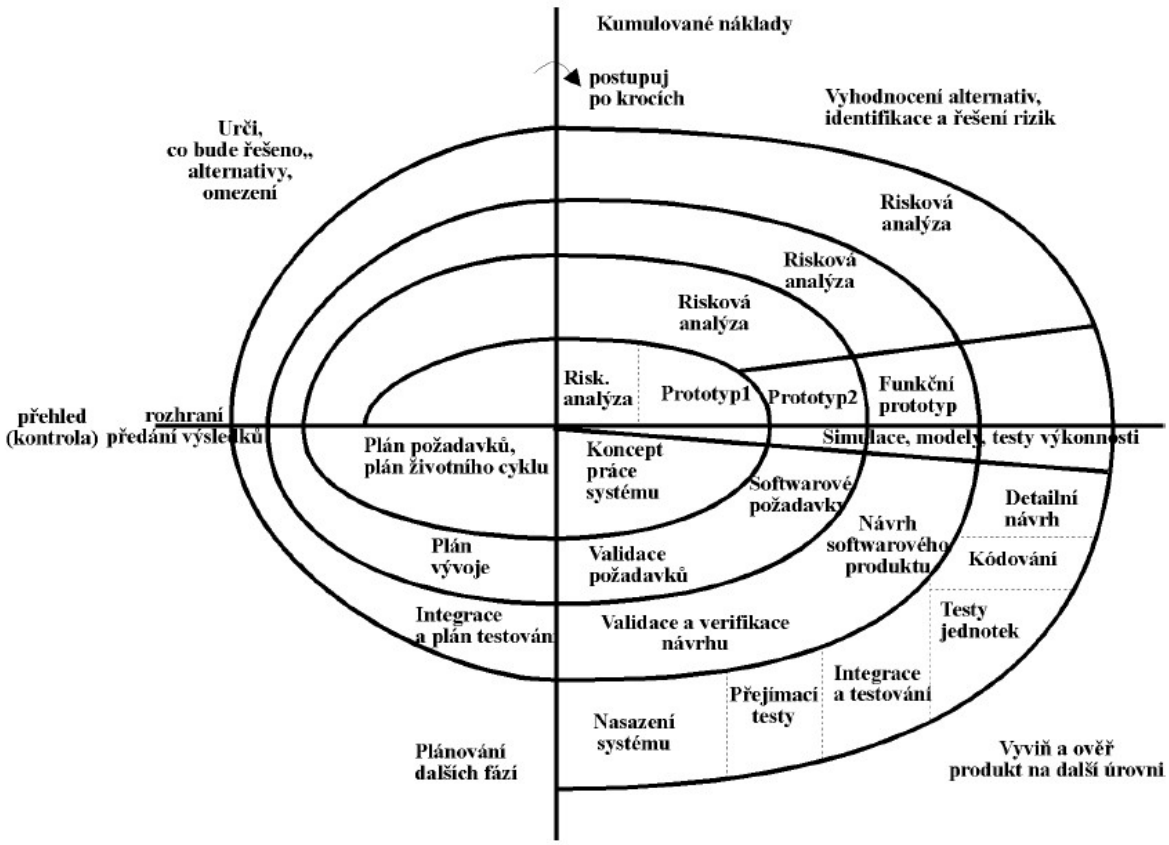
Posledním termínem, který patří do problematiky metodik, je nástroj. Nástroj „je prostředkem k uskutečnění určité činnosti v procesu vývoje a provozu IS a prostředkem k vyjádření výsledku této činnosti“ (Řepa a Chlapek, 1997, s. 21). Nástroj slouží jako softwarová podpora při vývoji softwarového produktu. Podle Hradecké (2008), příkladem nástroje je diagram toku dat DFD, diagram entit a jejich vztahu ERD, diagram hierarchické struktury, datový slovník atd.

2.5.2 Vývoj a rozdělení metodik

V současnosti se dají metodiky rozdělit do dvou hlavních proudů. Rigorózní nebo taky tradiční metodiky a agilní metodiky. V této kapitole se pokusíme tyto metodiky alespoň stručně popsat, protože podrobný popis a analýza metodik není předmětem této diplomové práce. Předtím, než se jim budeme věnovat důkladněji, si popíšeme jejich vývoj.

Každá metodika, která byla vytvořena, nějakým způsobem reagovala na problém aktuální doby. První metodiky byly reakcí na softwarovou krizi a na její důsledky. V některých případech nelze podle definice metodiky říct, že se jedná o metodiku, ale spíš o jakýsi model nebo schéma procesu vývoje softwaru. První takový model vznikl v 50. letech a měl pouhé tři etapy. Implementace, nasazení a odstranění chyb, případně rozšíření programu (Martinásek, 2011). V roce 1957 vznikl model Stagewise Model (v literatuře nepřekládán do českého jazyka), který již obsahoval většinu etap vývoje softwaru, které se používají i dnes. Byly to etapy: definice problému, specifikace požadavků, návrh architektury, implementace, integrace a provoz. V modelu, ale chybí zpětná vazba, testovaní a vyhodnocování úspěšnosti jednotlivých fází (Kadlec, 2004). V roce 1970 vznikl The Waterfall Model (Vodopádový model). Autorem modelu je Winstonu W. Royce a je základem většiny dnešních metodik, co se týče rozdělení fází. Model již obsahuje vyhodnocení jednotlivých fází (Kadlec, 2004).

V roce 1985 Barry Boehmen definoval Spiral Model (Spirálový model). Model reaguje na nedostatky vodopádového modelu a poprvé je v něm zahrnut i iterační přístup a důsledná analýza rizik. Model je postaven na myšlence, že každá fáze nemusí být provedena dokonale a může se opakovat. Dochází tedy k iteraci, na jejímž konci je vytvářen prototyp budoucího produktu. Spirálový model ovlivnil mnoho dalších modelů a procesů vývoje softwaru např. C-Bridge, Xerox Time-To Market Process, Ada Process Model nebo SPC Evolutionary Spiral Process. Ze spirálového modelu vychází dokonce i metodika RUP (Rational Unified Process), (Martinásek, 2011). Spirálový model, jak ho původně publikoval Barry Boehm (1988), vidíme na obrázku 2.2.



Obr. 2.2 Spirálový model. Obrázek je převzat od Suchora (2003)

V 80. letech začaly vznikat už komplexnější metodiky, mezi které patří například SSADM. V další dekádě začaly vznikat objektově orientované metodiky. Jejich hlavním představitelem je, již několikrát zmíněná metodika, Rational Unified Process. Další mezník ve vývoji metodik nastal začátkem 3. tisíciletí, kdy vznikly agilní metodiky, jako jsou XP, Crystal nebo Scrum. Podle Buchalcevové (2005a) rigorózní metodiky bývají zpravidla založeny na sériovém (vodopádovém) vývoji. Existují ale také rigorózní metodiky založené na iterativním a inkrementálním vývoji. I z tohoto důvodu budeme všechny metodiky, které nepatří do agilních metodik označovat metodikami rigorózními nebo též tradičními. Přehled metodik rozdělených do dvou skupin inspirované Kadlecem (2004):

Rigorózní – tradiční metodiky:

- UP (Unified Process),
- RUP (Rational Unified Process),
- EUP (Enterprise Unified Process),
- OPEN (Object-oriented Process, Enviroment and Notation).

Agilní metodiky:

- Scrum Development Process,
- Extrémní programování,
- Lean Development,
- Feature Driven Development,
- Test Driven Development,
- Crystal metodiky,
- Agile Modeling,
- Agile Unified Process,
- Adaptive Software Development,
- Dynamic Software Development Method.

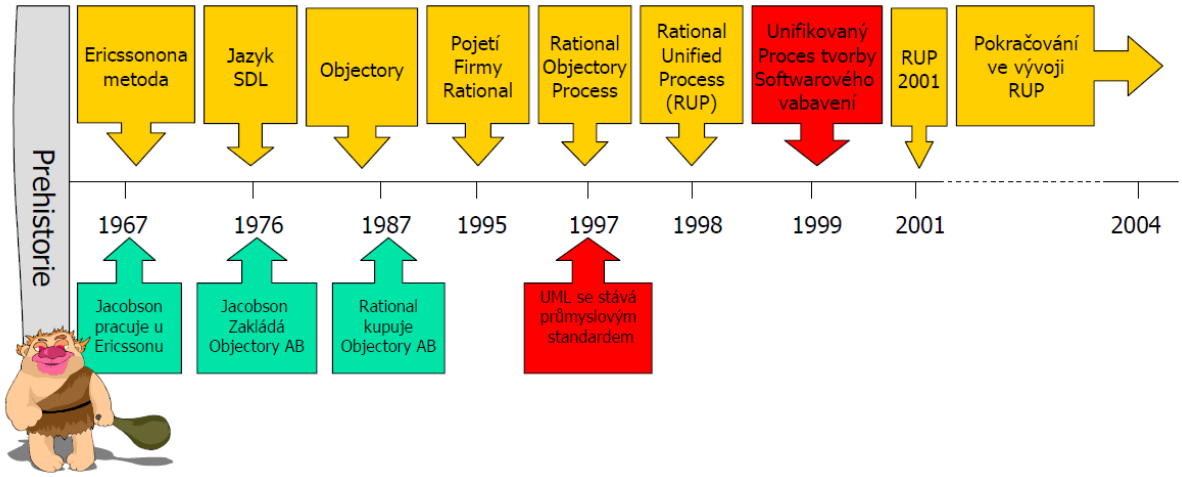
Jednoduché modely procesu vývoje softwaru jakou jsou vodopádový model nebo spirálový model nebudeme do rozdělení metodik zahrnovat. Důvody jsou jejich dnešní nevyužitelnost a fakt, že se ani o metodiky nejedná, (viz kapitola 2.6.1, kde jsme uvedli definice pojmů metodika a metoda). V následujících kapitolách si jednotlivé metodiky velmi stručně popíšeme.

2.5.3 Rigorózní metodiky

Mezi hlavní představitele rigorózních metodik patří metodika UP (Unified Process). Metodika vychází z metody Ericsson a Rational Objectory Process. Byla definována v roce 1999. Existuje několik komerčních variant této metodiky, které jsou potomky metodiky UP. Tyto metodiky dědí všechny její vlastnosti a v některých případech obsahují navíc různé nástroje, standarty, šablony apod. (Arlow a Neustadt, 2007).

„UP je vyspělý otevřený standart procesu tvorby softwarového vybavení vyvinutý autory UML“ (Arlow a Neustadt, 2007).

Vývoj a historie UP hezky dokládá obrázek 2.3 uveřejněn v knize o UML od autorů Arlow a Neustadt (2007).



Obr. 2.3 Vývoj metodiky UP (Arlow a Neustadt, 2007)

Mezi nejznámějšími metodiky vycházející z UP je metodika RUP (Rational Unified Process), která UP sice rozšiřuje, ale pracovní postupy a ideje jsou s UP totožné. Metodice RUP jako nejznámější rigorózní metodice se budeme zabývat nejvíce (Arlow a Neustadt, 2007). Dalšími metodikami, která vychází z metodiky UP jsou metodikx EUP (Enterprise Unified Process) a OPEN (Object-oriented Process, Enviroment and Notation).

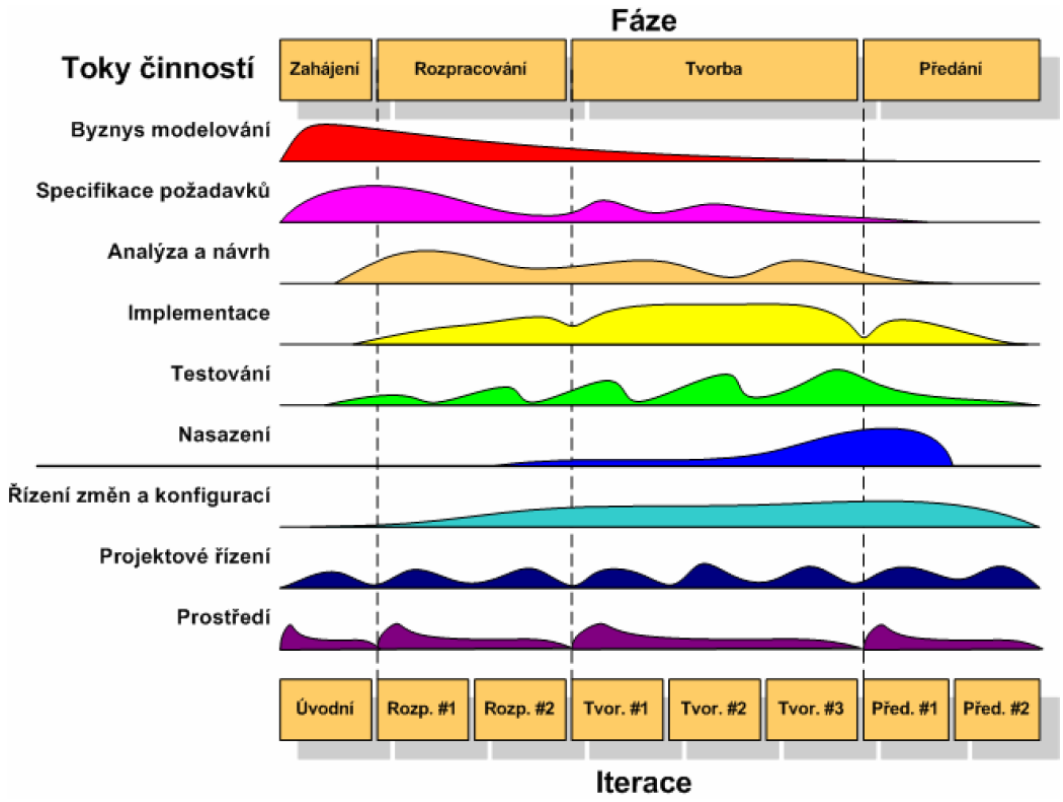
a) Rational Unified Process

Rational Unified Process je objektově orientovaná metodika s iterativním přístupem. Je velmi rozsáhlá a dopodrobna propracovaná, i to je důvod, proč je vhodná především pro velké firmy a pro složitější projekty. Právě kvůli své robustnosti může nasazení metodiky trvat delší dobu. Metodika vznikla v roce 1987 na základě metodiky Objectory Process, vyvinutou Ivarem Jacobsonem. Dnes metodiku vlastní firma IBM (International Business Machines Corporation), před ní však spadala metodika do firmy Rational, kde vznikala a odtud je i jméno metodiky. Součásti metodiky jsou i podpůrné nástroje, šablony, dokumenty a jiné podpůrné věci pro správný vývoj softwaru (Martinásek, 2011). Jako jedna z mála metodik, RUP řeší nejenom kdo, kdy a co, ale také jak. Základní čtyři prvky struktury procesu jsou následující.

- Pracovníky – jsou definovány, kompetence a zodpovědnost jednotlivých osob nebo skupin.
- Artefakty – reprezentují entity, které jsou v procesu vytvářeny, modifikovány a používány (modely, dokumentace, zdrojové kódy apod.).
- Činnosti – prováděné pracovníky s cílem vytvořit nebo upravit artefakty (kompilace zdrojových kódů, vytvoření návrhu apod.).

- Pracovní procesy – posloupnosti aktivit vytvářející požadované produkty (byznys modelování, specifikace požadavků, analýza a návrh, implementace, nasazení, testování, vyhodnocení (Martinásek, 2011).

Životní cyklus je tvořen čtveřicí fází: zahájení, rozpracování, tvorba a předání. V těchto fázích probíhají několik činností, které jsou zaznačený v obrázku 2.4, společně s objemem práce v jednotlivých fázích. Obrázek 2.4 zobrazuje vývoj softwaru v čase.



Obr. 2.4 Průběh vývoje SW dle metodiky RUP (Vondrák, 2002)

RUP obsahuje šest základních praktik, tzv. best practices (nejlepší praktiky), při vývoji softwaru, které při jejich dodržení mají vést k efektivnějšímu vývoji, lepšímu řízení kvality a lepším výsledkům. Nejlepší praktiky:

1. iterativní vývoj software,
2. správa požadavků,
3. architektura založená na komponentách,
4. vizuální modelování,
5. ověřování kvality software,
6. řízení změn software (Aldolf, 2008a), (Kadlec, 2003).

Těchto šest nejlepších praktik si popíšeme detailněji, protože je použijeme v praktické části této práce.

Iterativní vývoj software. Problém tradičního a sekvenčního přístupu je ten, že před sebou tlačí tušená i netušená rizika. A jejich odstranění je k blížícímu předání dražší a dražší. Důvodem je, že často nelze splnit všechny náležitosti každé fáze. Nelze přesně definovat všechny problému ve fázi analýzy. Navržené řešení ve fázi návrhu, taktéž nemusí být dokonalé. A implementace už vůbec ne. Iterativní přístup vychází ze spirálového modelu, kdy ke každé fázi se znovu vrací. V tomto iterativním a inkrementálním přístupu dochází k rozložení rizik průběžně. Každá z fází RUP obsahuje několik iterací, kdy výsledkem každé této interakce je spustitelná verze – prototyp. Další výhody iterativního vývoje softwaru jsou: již zmíněné odhalení rizik a nesrovnalostí mezi požadavky, návrhem a implementací, snazší správa změn, lepší znuvupoužitelnost, možnost testování meziproduktu, díky tzv. „prototypování“. To vede k většímu přiblížení k zákazníkovi a snazší plánování a řízení projektu.

Správa požadavků. Požadavky na systém jsou dynamické, mohou se v průběhu projektu měnit. Zákazník je často neschopný specifikovat všechny požadavky na systém na začátku projektu. Podléhá syndromu IKIWISI, což je akronym anglického výrazu I'll Know It When I See It (Já budu vědět, až když to uvidím). Tato zkratka je často používána v souvislosti s řízením a přípravou webových projektů. Popisuje chování klienta (zadavatele), který nedokáže odsouhlasit zadání připravené analytiky do té doby, než uvidí funkční aplikaci nebo web. Případně opačně, kdy klient zadání odsouhlasí, ale v okamžiku, kdy vidí výsledek, si uvědomí, co vlastně původně chtěl. (Kadlec, 2003). RUP proto zavádí aktivní správa požadavků během celého vývoje softwaru.

Architektura založená na komponentách. Je logické, že v případě produktu založeném na již použitých, funkčních a otestovaných komponentách, se dá ušetřit podstatnou část zdrojů. Projekt lze navíc díky tomu rozdělit mezi několik vývojových týmů a systém vyvíjet souběžně. Proto RUP nabízí metodickou, systematickou cestu návrhu, vývoje a ověření správnosti architektury zahrnující šablony, architektonické styly a pravidla designu.

Vizuální modelování. Model je zjednodušení reality za účelem lepšího porozumění systému, zvlášť u větších projektů je velmi těžké pochopit systém v celé jeho velikosti. Proto RUP využívá modelovací jazyk UML, který mu umožňuje zjednodušení komunikace v týmu, zlepšení konzistence projektu a jak již jsme uvedli, zvýšení pravděpodobnosti správného pochopení systému.

Průběžné ověřování kvality softwaru. Aby software mohl být nasazen, musí dle RUP splňovat tři kvalitativní kritéria: Funkcionalita: systém obsahuje všechny funkce uvedené v modelu případů užití. Spolehlivost: systém pracuje spolehlivě, řeší všechny možné chybové stavy. Výkon: systém je permanentně dostupný a doby odezvy jsou přijatelné.

Řízení změn software. Neřízení změn v jakékoliv fázi projektu mohou vést k chaosu a k velkým problémům. Změny proto musí být řízeny a dokumentovány.

Pro některé vývojové týmy může být problematické, že pro každý nový projekt je třeba vytvořit novou instanci metodiky RUP. Pro každý nový projekt se musí RUP nejdříve přizpůsobit, aby metodika byla vůbec použitelná. To má za následek vynaložení času a zdrojů navíc.

b) Enterprise Unified Process

Autorem metodiky EUP je Scott W. Ambler a rozšiřuje metodiku RUP v několika ohledech. Za prvé pokrývá procesy na úrovni celé organizace. Definuje Infrastructure Management zahrnující procesy realizované napříč projekty. Připojuje fáze Production (Produkce), obsahující provoz a údržbu systému a fázi Retirement (Výslužba), popisující činnosti pro stražení produktu z používání (Buchalcevová, 2005a).

c) Obejct-oriented Process, Environment and Notation

Metodika OPEN vychází z projektu COMMA (Common Object Methodology Metamodel Architecture), který měl cíl vytvořit z několika již definovaných metodik, univerzální metodiku pro objektově orientovanou analýzu a návrh. OPEN je tedy dílem několika desítek vědeckých pracovníků a vývojářů. Největší zásluhy jsou však přisuzovány autorům Brianovi Henderson-Sellersi, Meilir Page Jonesovi, Ianovi Grahemovi a Donaldovi Firesmithovi. OPEN je veřejně přístupná metodika, která slouží pro celý cyklus vývoje softwaru. Definuje procesní rámec OPF (OPEN Process Framework), který se vyznačuje svou univerzálností (Buchalcevová, 2005a).

2.5.4 Agilní metodiky

Na začátku 3. tisíciletí si začalo několik předních softwarových inženýrů uvědomovat, že v jistých případech přestávají tradiční a rigorózní metodiky efektivně fungovat. To je vedlo k založení neziskové organizaci The Agile Alliance, která měla propagovat agilní postupy a zásady při vývoji softwaru.

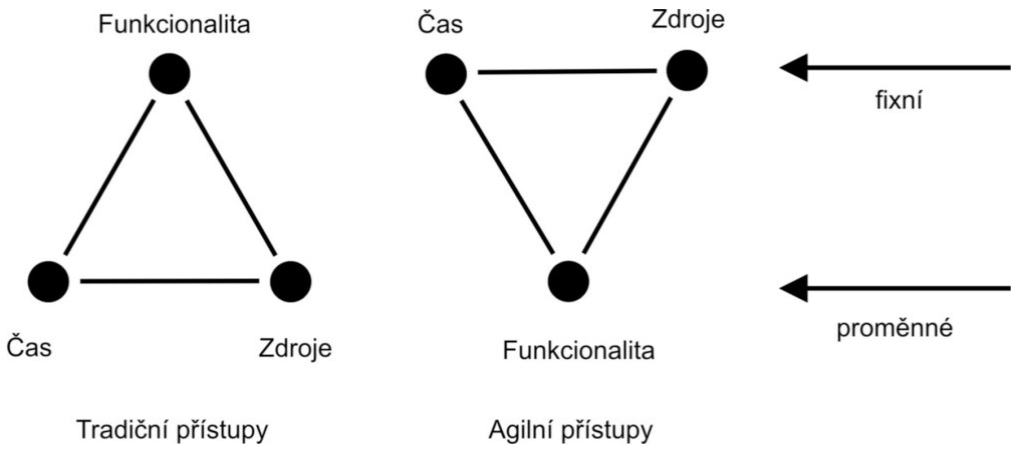
V roce 2011 tyto agilní zásady byly definovány v manifestu The Agile Manifesto[1]. Inspirací překladu z anglického originálu (Beck et al., 2001a) nám bude překlad z mé bakalářské práce (Martinásek, 2011) a překlad Michala Valloa ve spolupráci členů agilní komunity Agília (Vallo et al., 2001).

1. Nejvyšší priorita je uspokojit zákazníka díky včasné a průběžné dodávce softwaru.
2. Vítají se požadavky na změnu, a to i na konci vývoje softwaru, změna může být pro zákazníka konkurenční výhodou.
3. Průběžně a pravidelně dodávat klientovi funkční software. V intervalech od několika týdnů po měsíce, snaha je ale co nejkratší interval.
4. Lidé z byznysu a vývojáři musí pracovat spolu denně a to v průběhu celého projektu.
5. Projekt by měl být stavěn na motivovaných jedincích. Vedení musí podporovat potřeby těchto jedinců, důvěřovat jim a zajistit jim prostředí pro perfektní práci.
6. Nejúčinnější a nejefektivnější způsob předávání informací v rámci vývojového týmu a komunikace mimo něj je osobní konverzace.
7. Postup práce se měří fungujícím softwarem, který je brán veličinou pro toto měření.
8. Agilní procesy podporují trvale udržitelný rozvoj. Všichni, to znamená, sponzoři, vývojáři a uživatelé, by měli udržovat konstantní tempo rozvoje.
9. Neustálá pozornost na kvalitní návrh a technické dokonalosti rozvíjejí agilitu a zvyšují tím rychlost procesu vývoje.
10. Jednoduchost, efektivita a zaměřit se na opravdu nejdůležitější věci tvoří základ.
11. Samo-organizující týmy mají nejlepší výsledky.
12. V pravidelných intervalech by se mělo vyhodnocovat, jak pracovat ještě efektivněji a jak upravit a zlepšit svůj způsob práce.

Jedna z nejzápadnějších podmínek agilních metodik je, že pracují s iteracemi, které by měly být velmi krátké. Jak napovídá název agilní (hbitý), vývoj softwaru musí být rychlý a svižný. Aby se dosáhlo spokojenosti klienta s vyvíjeným softwarem, je mu průběžně představován ve formě prototypů a získání tímto způsobem zpětnou vazbu. A to nejčastěji jak je to možné. Na tomto principiu můžeme vidět další rozdíl mezi agilními a tradičními metodikami, a to velmi úzkou spolupráci se zadavatelem projektu.

[1] Autoři manifestu jsou Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas (Beck et al., 2001b).

U tradičních metodik hrají hlavní roli požadavky klienta, tedy funkcionalita, která je fixním faktorem. Cena a termín dodání jsou vedlejší a berou se jako proměnné. U agilních metodik je to přesně naopak. Fixními faktory jsou cena a čas, které se stanoví na začátku projektu. A požadavky na funkcionalitu se může v průběhu vývoje měnit. Zachycení těchto rozdílu zobrazuje obrázek 2.5.



Obr. 2.5 Rozdíl tradičních a agilních přístupů (Kadlec, 2004)

Propagátoři agilních metodik si dále uvědomili, že hlavním faktorem při vývoji softwaru jsou lidské zdroje. Proto je u agilních metodik největší důraz kladen na spolupráci a komunikaci mezi členy vývojového týmu. A to především osobní formou kontaktu, tedy formou, která je v dnešním digitálním světě potlačována.

Autoři kromě zmíněných dvanácti zásad ve svém manifestu (Beck et al., 2001b) také uvádějí, které hodnoty upřednostňují před hodnotami tradičního vývoje softwaru. V agilním manifestu přeloženém Vallo et al., (2001b), autoři uvádějí, že dávají přednost:

- individualitě a interakci před procesy a nástroji,
- fungujícímu softwaru před obsáhlou dokumentací,
- spolupráci se zákazníkem před sjednáváním smluv,
- reakci na změny před plněním plánu.

Tradiční rigorózní přístupy pohlížejí na vývoj softwaru jako na inženýrskou disciplínu. Alistair Cockburn, autor agilního manifestu, ke kterému se dostaneme později, se však dívá na vývoj software jako na kooperativní hru s omezenými zdroji založenou na invenci a komunikaci (Buchalcevová, 2005a). Jako příklad kooperativní hry uvádí tým horolezců. Hra horolezců má jasný konkrétní cíl, je kooperativní a konečná. Stejně je to s vývojem softwaru. Primárním cílem při tvorbě software je dodání softwaru, který splňuje požadavky zadavatele.

Sekundárním cílem je pak připravit se pro další hru, kterou je rozšíření softwaru nebo vývoj nového. Pokud v primárním cíli selžeme, je sekundární cíl bezpředmětný. Stejně tak dodávat podrobnou dokumentaci, když nedodáme fungující software. Avšak úspěšné splnění primárního cíle nemusí znamenat také úspěch v sekundárním cíli, tedy úspěch v dalších hrách (Buchalcevová, 2005a).

Agilní metodiky jsou přizpůsobené spíše pro menší vývojové týmy. Počet členů v týmu by se měl pohybovat mezi dvacet až čtyřicet lidí, největšího výkonu, ale firmy dosahují s deseti členy (Buchalcevová, nedatováno).

Na následujících stranách si agilní metodiky podle Kadlece (2004) velmi stručně popíšeme. Pokud nebude uvedeno jinak, popsání metodik bude převzato z mé bakalářské práce (Martinásek, 2011). Metodiku Scrum jako základní teoretický prvek této práce pak budeme popisovat v samostatné kapitole.

a) Extrémní programování

Extreme Programming (Extrémní programování) je velmi populární metodika, která je známá především svou extrémností (jak plyne z názvu) a různými zvláštnostmi. Základem je dodávání nových verzí softwaru průběžně. Šetří čas při tvorbě dokumentace a specifikací. Dokumentace se prakticky nepíše. Metodiku navrhl známý softwarový inženýr Kent Beck, (2002) v roce 1999. A jak název metodiky napovídá, metodika propaguje vše, co se ukáže jako prospěšné dotahovat do extrému. Filosofie XP je položena na pěti hodnotách: komunikace, jednoduchost, zpětná vazba, odvaha a respekt. Vývoj na základě XP (Extreme Programming – Extrémní programování) je zábavný a v jistém smyslu hravý, je však podmíněn správným složením týmu. Je to pružná a efektivní metodika, která je vhodná spíš pro věkově mladé týmy než pro týmy, které jsou již naučeny myslet tradičně (Kadlec, 2004). Počet členů týmu by měl být mezi dvěma až deseti.

b) Lean Development

Počátky metodiky Lean Development (někdy nazývaná Lean Software Development) začal definovat Robert Charette v 80. letech minulého století. Inspirací mu byly principy Lean Manufacturing a Total Quality Management. Principy, které byly nejdříve uplatňovány v automobilovém průmyslu v Japonsku (Buchalcevová, 2005a). I proto metodika nepopisuje jak vyvíjet nebo řídit softwarový proces, ale definuje pouze deset základních pravidel, které byly sice ohnuty pro zefektivnění vývojového procesu, ale mohou být využity v jakémkoliv výrobním procesu. Myšlenkou této metodiky je odstranění zbytečných úkolu, jako je například psaní dokumentace, kterou nebude nikdo číst nebo zabývání se funkcemi systému,

které nebude nikdo používat. (Kadlec, 2004) (Poppendieck M. a Poppendieck T., 2003) Cíl metodiky dobře definuje Buchalcevová (2005a): „Cílem Lean Development je vytváření softwaru tolerantního ke změnám a třetinovou lidskou prací, s třetinovým časem, s třetinou investic do nástrojů a metod, s třetinovou námahou přizpůsobit se novému tržnímu prostředí“.

c) Feature Driven Development

Metodika FDD (Feature Driven Development – Vývoj řízený vlastnostmi) vznikla v 90. letech a její autoři jsou Jeff De Luca a Peter Coad. Jak poukazuje název metodiky, základním elementem je zde vlastnost, konkrétně vlastnost systému. Na začátku procesu se vytvoří setříděný seznam funkcí – vlastností, podle priorit a podle kterého se pak v interacích, kde je prezentován zákazníkovi, vyvíjí. Metodika klade důraz na modelování a přesné pochopení vlastností systému, které klient požaduje. Pro každou vlastnost nebo skupinu vlastností se může sestavit nový speciální tým.

Metodika je doporučená a vhodná především pro menší týmy a projekty. V situacích, kde je nutné rychlé dodání softwaru, není moc vhodná, protože tým se může během tvorby softwaru dynamicky měnit a tím může vývoj softwaru zpomalovat (Kadlec, 2004).

d) Test Driven Development

Metodika Test Driven Development (Vývoj řízený vlastnostmi), označována pod zkratkou TDD se odlišuje od ostatních metodik a vlastně i procesu vývoje jako takového, že vývoj začíná nejdříve testováním a až pak implementace. Jako první se napíše test pro danou funkci nebo problém a až posléze se začíná implementovat. Tento reversní přístup k vývoji softwaru, zajišťuje vysokou kvalitu softwaru. Jelikož to, co není otestováno, není ani naprogramováno a součástí systému. Test Driven Development se příliš nezabývá tvorbou specifikace, dokumentace. Metodika vyžaduje velmi zkušené programátory, kteří musí být současně i testeři a zároveň striktní a direktivní vedení (Kadlec, 2004), (Beck, 2002).

e) Crystal metodiky

Crystal metodiky je rodina metodik, která se dělí na další skupiny rozdělné podle společných pravidel a principů. Autorem Crystal metodik je Alistair Cockburn (Cockburn, 2004). Základní skupina se jmenuje čistý Crystal Clear. Podskupiny jsou pak pojmenovány podle barev. Yellow Crystal (žlutý), Red Crystal (červený) atd. Stejně jako u Lean Development i zde metodika doporučuje se nevěnovat zbytečným věcem. A tak jako u většiny agilních metodik se snaží vyhnout dokumentování a papírování. Naopak je kladen důraz na lidský faktor.

Čím se liší od ostatních agilních metodik, je myšlenka, že každý vývojový proces je jiný. Proto je Crystal pouze jakási šablona, která se na začátku každého projektu musí nejdříve upravit a přizpůsobit konkrétnímu procesu. Uzpůsobení metodiky pro každý projekt může být ale v některých případech kontraproduktivní a může prodlužovat vývoj softwaru (Kadlec, 2004).

f) Agile Modeling

Agile Modeling (agilní modelování), dříve známe pod názvem Extreme Modeling (extrémní modelování) považuje většina autorů jako metodiku. Často se používá pouze jako doplněk k jiným agilním metodikám (Kadlec, 2004), (Ambler, 2012). Podle Buchalcevové (2005a), může být proto agilní modelování začleněno jak do tradičních metodik, tak do metodik agilních. Agile Modeling, jak naznačuje název je totiž jen soubor hodnot, principů a postupů pouze pro modelování, případně pro dokumentaci, které vychází z metodiky XP.

Principy a praktiky agilního modelování jsou vyobrazeny v příloze A, tabulky jsou převzaty od Buchalcevové (2005a).

g) Agile Unified Process

Metodika AUP (Agile Unified Process) vychází z metodiky RUP, principy však přebírá z agilního vývoje, proto se řadí mezi agilní metodiky a ne do skupiny rigorózních. Metodika vznikla v roce 2005 spojením metodik a RUP a XP (Leffingwell, 2011). Cyklus vývoje se shoduje s metodikou RUP. Výhoda je však, že na rozdíl od RUP, je AUP volně přístupná.

h) Adaptive Software Development

Autorem ASD (Adaptive Software Development) je Jim Highsmith. Základem je myšlenka adaptace na změny. Proto tradiční fáze plánování, návrh, realizace jsou vyměněny za fáze: Speculate (Spekulace), ve které se provádí vše potřebné pro zahájení projektu, Collaborate (Spoluprace) obsahující činnosti pro samotný vývoj softwaru a Learn (Učení), kde se provádí zhodnocení projektu ze všech možných úhlů pohledu. Díky tomuto rozdělení je pro tým snazší se přizpůsobit na všechny možné změny v cyklu vývoje softwaru (Buchalcevová, 2005a).

i) Dynamic Software Development Method

Metodika DSDM (Dynamic Software Development Method) vznikla v roce 1995 ve Velké Británii, konkrétně v konsorcii DSDM Consortium. Metodika vychází z RAD (Rappid Application Development – rychlíý vývoj aplikací). Metodika je velmi kvalitně dokumentována stejně tak systém školení na její osvojení (Buchalcevová, 2005a). DSDM je

postaveno na osmi principech, které velmi podobají obecným agilním principům a nebudeme si je zde uvádět (Kadlec, 2004). DSDM definuje podle Buchalcevové, (2005a), čtyři fáze procesu vývoje. První fází je Study (Stuide), rozdělná na Feasibility Study (Studium proveditelnosti) a Business Study (Byznys studie). Následují fáze Funcional Model (Funkční model), Design and Build (Návrh) a Implementation (Implementace). DSDM předepisuje složení týmu, zavádí kategorizaci požadavků podle priorit a časové intervaly pro realizaci skupiny požadavků. Metodika je bohužel pod placenou licencí (Kadlec. 2004).

2.5.5 Výběr metodiky

Zásadní otázkou, týkající se metodik, kterou si každý projektový manažer klade, je, jakou metodiku pro vývoj daného softwaru vybrat pro daný projekt. Tak jako každý projekt vývoje softwaru, tak i každá metodika se nějakým způsobem liší. Je proto logické, že na různý problém – projekt je vhodná jiná metodika. Bylo by krajně neefektivní například na vývoj jednoduché aplikace používat některou z robustních metodik jako například Rational Unified Proccess. Je evidentní, že firma nebude na každý projekt používat a hledat jinou metodiku, ale seskupí projekty podle určitých kritérií a zvolí jednotnou metodiku pro všechny podobné projekty.

Kritéria, podle kterých firma muže hledat optimální metodiku podle Buchalcevové (2005a) jsou následující.

1. Různé technologie vyžadují různé techniky a metody.
2. Objektově orientované metodiky vyhovují projektům, které mají základ v objektově orientované technologii, datově orientované metodiky vyhovují pro vývoj datově orientovaných aplikací apod.
3. Organizace se liší firemní kulturou. Mnohé implementace metodik mohou selhat, protože nepočítají s firemní kulturou. Různé přístupy podporují jinou firemní kulturu. Před implementací metodiky do organizace je třeba provést analýzu její firemní kultury.
4. Každý tým a jedinec je jedinečný.
5. Velikostí týmu na projektech se liší. Pro malý tým obvykle stačí menší metodika a naopak, pro větší tým je ideální větší metodika.
6. Projekty se liší svou důležitostí a složitostí. Software pro nemocnice a herní software vyžadují jiné metodiky z důvodu své důležitosti. Metrik pro jejich měření je mnoho.
7. Pro softwary vstupující na trh poprvé se zpravidla používají velmi malé metodiky. Naopak pro software, již zavedený na trhu, se používá některá z rigorózních metodik. Produkt,

pro který je důležitý rychlý vývoj kvůli své konkurenční výhodě (zvlášť v prostředí internetu to platí dvojnásob) potřebuje některou z agilních metodik.

8. Některé projekty musí odpovídat různým normám a pravidlům pro státní zakázky, některé projekty mají vazbu na dodavatele a jejich požadavky. Mají přesně stanovené zdroje a termíny.

Nalezení odpovídající metodiky pro projekt a firmu je alfou a omegou problematiky vývoje softwaru. Předmětem této diplomové práce však není nalezení optimální metodiky, ale nalezení hlavní příčiny selhání a optimalizovat nasazení již vybrané metodiky Scrum do firmy.

2.5.6 Stav v oblasti metodik v České republice a ve světě

To, jak jsou metodiky důležité pro správný a efektivní vývoj softwaru, jsme si již uvedli na začátku této práce. Jaký je, současný stav využívání metodik u nás a ve světě, si popíšeme v této kapitole. Vycházet budeme z (Buchalcevová, 2006) a (Buchalcevová, 2005a).

Dle studie společnosti Standish Group v roce 2000 splňovaly kritéria úspěšnosti[2] pouze z 28 %. Což není vůbec lichotivá statistika a odráží potřebu a význam používání metodik. Společnosti META Group, provedla průzkum, ze kterého vyplývá, že 51,6 % oslovených společností používá při vývoji nějakou metodiku. V ČR je podle Buchalcevové (2006) procento používání metodik ještě daleko nižší.

Hlavní vinou, proč ČR zaostává za světem, je podle Buchalcevové (2005a), nedostatek českých metodik. Tím podle ní vzniká jazyková bariéra, která zároveň znesnadňuje metodiku v cizím jazyce aplikovat. Další vina je nedostatek finančních prostředků na nákup komerčních metodik a uvolnění lidských zdrojů na problematiku optimalizace vývoje.

Faktory neúspěchu u těch, kteří metodiku používají, ale i tak mají špatně výsledky, jsou, že metodiky nejsou jednotně popsány, nejsou kategorizovány a velmi špatně se pak z nich vybírá optimální metodika na daný problém. Navíc některé metodiky se zaměřují jen na určitou fázi vývoje nebo typ projektu. Dále nejsou definovány postupy pro výběr metodiky ani pro její správnou implementaci a přizpůsobení na konkrétní projekty a firmy.

[2] Kritéria úspěšnosti: byl projekt odevzdán v dohodnutém termínu? Byl dodržen rozpočet? Byly dodrženy všechny požadované specifikace na systém?

2.6 Scrum

2.6.1 Úvod

Slovo Scrum je z anglického výrazu pro mlýn ve sportu ragby a často je překládáno jako „skrumáž“ (Kadlec, 2004). Metodika Scrum je jedna z nejpopulárnějších agilních metodik. Celým názvem Scrum Development Process, je v literatuře a běžně v praxi označována pouze jako Scrum[3].

Metodiku Scrum využívají přední světové firmy, jako jsou Microsoft, Yahoo, Google, Philips, Siemens, Nokia, Lockheed Martin nebo BBC (Mountain Goat Software, 2012). Mezi české firmy, které používají Scrum patří například seznam.cz (Bobek, 2012), firma LMC provozující portály prace.cz a jobs.cz (Itbiz, 2011) a ESET ČR (Knesl, 2010).

Autoři Scrumu jsou Ken Schwaber a Jeff Sutherland (Schwaber a Sutherland, 2012). Jeff Sutherland vytvořil první Scrum tým už v roce 1993 a spolu s Kenem Schweberem v roce 1995 metodiku Scrum publikovali v příspěvku Business Object Design and Implementation Workshop, na konferenci OOPSLA'95 (Object-Oriented Programming, Systems, Languages & Applications '95) v Americe. Po té jej začali aplikovat ve firmách (Schwaber a Sutherland, 2007). Dalším autorem Scrumu je považován Mike Beedle, který společně se Schwaberem popsali metodiku Scrum v knize Agile Software Development with Scrum (Schwaber a Beedle, 2001). Všichni tři se také podíleli na sepsání Agile Manifesto v roce 2001 (Beck et al., 2001b).

Scrum je proces sloužící pro agilní vývoj softwaru. Zatímco ostatní agilní metodiky mohou být většinou použity na jakýkoliv i ne softwarový projekt, Scrum se zejména hodí pro projekty s rychle se měnícími a naléhavými požadavky, které můžeme nalézt u vývoje softwaru a to především v prostředí internetových zakázek (Cohn, 2012m). Scrum je jednoduchý, srozumitelný, ale extrémně obtížný pro dokonalé zvládnutí (Schwaber a Sutherland, 2011).

[3] V literatuře, je někdy název metodiky psát velkými písmeny SCRUM, což evokuje, že se jedná o akronym.

2.6.2 Slovník metodiky Scrum

V informatice je běžné používat anglické názvy. Asi proto se v české literatuře při popisu metodiky Scrum používají někdy české a někdy anglické termíny. Což může čtenáře mást. V mnoho případech u některých slov totiž ani neexistuje český ekvivalent.

Aby tedy nedocházelo k nejasnostem, při popisu metodiky Scrum je v tabulce 2.1 uveden výčet všech použitých anglických výrazů a jejich český ekvivalent[4], který, bývá používaný v české literatuře.

Název v originálu	**Český ekvivalent**
Artifacts	artefakty
Burndown Chart	burndown graf
Ceremonies	schůzky
Daily Scrum Meeting	denní scrum schůzka
Planning Poker	plánování pokerem
Product Backlog	produktový backlog
Product Owner	vlastník produktu
Release Burndown chart	burndown graf uvolnění
Roles	role
Scrum	Scrum
Scrum team	scrum tým
ScrumMaster	ScrumMaster
Sprint	sprint
Sprint Backlog	sprint backlog
Sprint Burndown chart	sprint burndown graf
Sprint Planning	plánování sprintu
Sprint Retrospective	retrospektiva sprintu
Sprint Review	hodnocení sprintu
Team	tým
Time-box	časový rámec
User Stories	uživatelské příběhy

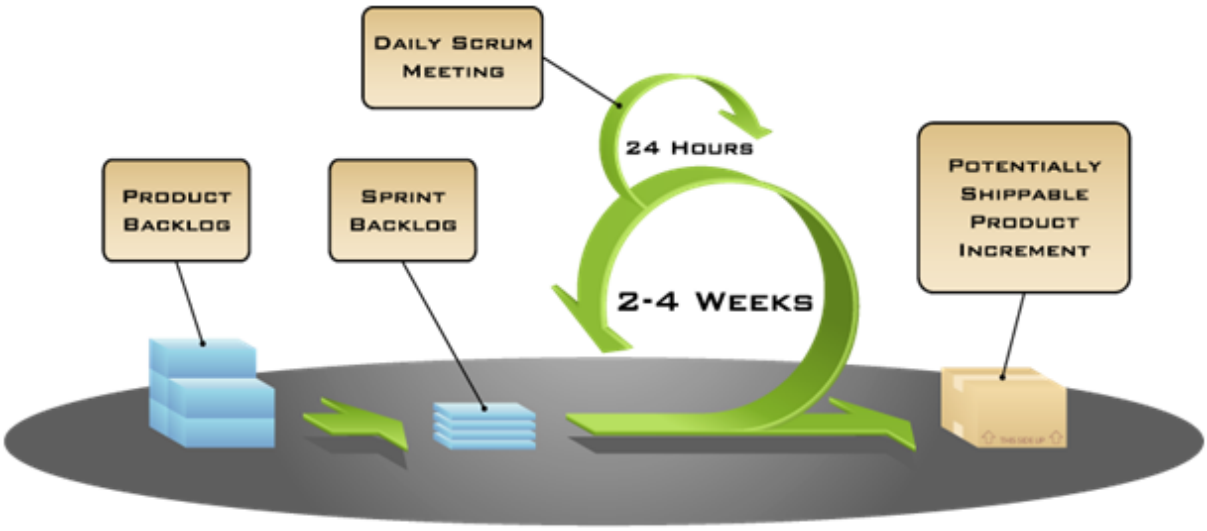
Tab. 2.1 Termíny metodice Scrum a jejich české ekvivalenty

[4] V některých případech nejde o doslovný překlad, ale o překlad vystihující podstatu věci nebo překlad již zaběhnutý v informační praxi.

2.6.3 Teorie metodiky Scrum

I když je metodika Scrum brána jako metodika, Scrum nepopisuje detailně a komplexně co se má při vývoji udělat. To nechává na vývojovém týmu, který ví, jak problémy řešit (Cohn, 2012m). Scrum je tedy procesní rámec, do kterého lze zakomponovat jiné procesy, techniky a nástroje, ale i další metodiky (Schwaber, 2010). Nejčastěji bývá Scrum spojen s metodikou XP (Kniberg, 2007).

Scrum se skládá z Scrum týmů a přidružených rolí, činností, artefaktů a pravidel. Tyto sady pravidla podporují inovaci, kreativitu, komunikaci, růst a učení. Scrum je založen na principu zkontroluj a uprav (Schwaber a Sutherland, 2011). Scrum je de facto proces, rozdělený na malé časové díly tzv. Timeboxy (časové rámce). Scrum se skládá s řady iterací zvané sprinty. Každý sprint trvá dva až čtyři týdny. Tento sprint lze dále rozdělit na menší denní iterace (Schwaber a Sutherland, 2011). Životní cyklus podle metodiky Scrum je znázorněn na obrázku 2.6.



Obr. 2.6 Životní cyklus podle metodiky Scrum (Cohn, 2012m)

„Scrum je založen na teorii řízení empirických procesů[5] neboli empirismu. Empirismus tvrdí, že znalost pochází ze zkušenosti a rozhodování založeném na tom, co je známo. Scrum využívá iterační a inkrementální přístup k optimalizaci předvídatelnosti a řízení rizik. Každá implementace řízení empirického procesu stojí na třech pilířích: transparentnosti, kontrole a adaptaci“ (Schwaber a Sutherland, 2011, s. 3).

[5] Kromě empirického procesu existuje také definované řízení procesu. Tedy takové řízení procesu, které je fixní a řízení procesu se v průběhu tvorby neupravuje. A bývá jasná kvalita a cena výsledného produktu (Schwaber, 2004).

- **Transparentnost.** Aspekty procesu musejí používat společný standard, aby byly srozumitelné a viditelné pro všechny, kteří mohou ovlivnit výsledek. Účastníci například musejí používat společný jazyk pro popis procesů nebo přesně definovat, kdy je úkol splněný (Schwaber a Sutherland, 2011).
- **Kontrola.** Aby se nalezly odchylky v procesu, Scrum tým musí kontrolovat artefakty a postup vedoucí k cíli s optimální frekvencí. Tato frekvence nesmí být natolik vysoká, aby neúměrně zdržovala v práci. Aby byla kontrola nejužitečnější, musejí ji provádět kvalifikování lidé, a to přímo na místě, kde se pracuje (Schwaber a Sutherland, 2011).
- **Adaptace.** Pokud se při kontrole nalezne alespoň jeden bod v procesu mimo hranici požadavků, nebo se zjistí, že výsledný produkt je nepřijatelný, musí se provést adaptace procesu. Napravení této odchylky se musí provést okamžitě, aby nenapáchala v budoucnu další škody. Scrum definuje čtyři formální body pro kontrolu a adaptaci: Sprint Planning, Daily Scrum, Sprint Review a Sprint Retrospective[6] (Schwaber a Sutherland, 2011).

Základní prvky metodiky Scrum podle Schwaber a Sutherland (2007) jsou:

- role,
    - vlastník produktu,
    - ScrumMaster,
    - vývojový tým,
- schůzky,
    - plánovací schůzka,
    - vyhodnocení sprintu,
    - retrospektiva sprintu,
    - denní Scrum schůzka,
- artefakty,
    - produktový backlog,
    - sprint backlog,
    - burndown grafy[7].

[6] Viz v popisu činností ve Scrumu, kapitola 2.7.5 Činnosti.

[7] Strukturu podkapitol popisující jednotlivé prvky rámce nám bude inspirací výše uvedené složení Scrumu. Pouze schůzky přejmenujeme na kapitolu činnosti, protože do ni přidáme i popis samotného sprintu a techniku časového ohodnocování položek v baklogu tzv. Planning Poker (Plánování pokerem).

2.6.4 Role

Scrum tým se skládá z členů, kteří zastávají role: Product Owner (vlastník produktu), ScrumMaster a Team (tým) (Schwaber a Sutherland, 2011).

a) Vlastník produktu

Zodpovídá za nejvyšší možnou kvalitu produktu a práce vývojového týmu. Vlastník produktu odpovídá za správu produktového backlogu. Vlastník produktu má podle autorů Schwabera a Sutherlanda (2011) na starosti tyto činnosti:

- definování položek v produktovém backlogu,
- uspořádání těchto položek dle vize produktu a cíle,
- kontrola hodnot a zajištění jejích dodání,
- zajištění transparentnosti a dostupnosti produktového backlogu, aby každý člen týmu byl informován, na čem se bude v nejbližší době pracovat,
- zajištění aby vývojový tým správně tyto položky pochopil.

Vlastník produktu je dále zodpovědný za ROI[8] respektive za jeho maximalizaci. Vlastník produktu je konkrétní osoba, která zodpovídá za tyto činnosti, může jimi, ale pověřit někoho jiného. Musí být respektován vývojovým týmem. Je nezpochybnitelnou autoritou, nikdo jiný nesmí upravovat backlog bez jeho vědomí a souhlasu (Schwaber a Sutherland, 2011).

Ve vývojovém procesu zastupuje zákazníka, představuje jeho potřeby a zodpovídá za samotný projekt (Schwaber a Sutherland, 2007). Kvůli této velmi zásadní úloze si následně popíšeme čtyři základní požadavky na dobrého vlastníka produktu (Sutherland, 2013).

- Informovanost – vlastník produktu musí znát trh, zákazníka, produkt i konkurenci. V opačném případě může vlastník produktu ztratit důvěru týmu a to může mít za následek zpomalení vývoje produktu. Aby vlastník produktu měl všechny tyto znalosti, potřebuje pomoc a podporu od vedení společnosti.
- Dostupnost – Správný vlastník produktu by měl být napůl obchodník. Polovičku své pracovní doby by měl trávit se zákazníkem a tu druhou s vývojovým týmem. Musí být oběma skupinám neustále dostupný.

[8] "ROI je zkratka z anglického Return On Investments, tedy návratnost investic. Jako ROI (někdy též ROI index) označujeme poměr vydělaných peněz k penězům investovaným. ROI tedy udává výnos v procentech z utracené částky. ROI (%) = výnosy / investice * 100" (Adaptic, 2013).

- Rozhodování – vlastník produktu má konečné slovo, co se týče produktového backlogu. Musí být nezpochybnitelnou autoritou, být rozhodný. Pokud nejsou tyto podmínky splněny, vede to ke konfliktům a zpomalení práce.
- Odpovědnost – vlastník produktu je odpovědný za obchodní plán a je zodpovědný za zisk. Přepočítává položky z produktového backlogu na příjmy, tak aby dokázal vyčíslit část práce v penězích.

Jmenován vlastníkem produktu bývá často produktový manažer, obchodník nebo někdo z marketingového oddělení (Cohn, 2012d).

Podle Roman Pichler (2010) je vlastník produktu jednou z nejdůležitějších osob. De facto je zodpovědný za finální úspěch projektu, spokojenost klienta a tedy výše zisku. Proto se Pichler ve své knize Agile Product Management with Scrum: Creating Products that Customers Love zabývá, jak pomocí metodiky Scrum dosáhnout nejlepšího výsledku právě díky vlastníka produktu a produktového backlogu.

b) ScrumMaster

ScrumMaster není vedoucí Scrum týmu v pravém slova smyslu. Je pouze zodpovědný za učení a dodržování Scrum pravidel Scrum. Kontroluje, zda jsou dodržovány techniky, pravidla a idea Scrumu (Schwaber a Sutherland, 2011). ScrumMáster je brán jako trenér, učitel týmu, který pomáhá používat rámec Scrum (Deemer et al., 2012). Učí nové členy týmu pochopit jejich roli a ukazuje jim směr, aby byli maximálně prospěšní týmu. Na dalších řádcích si ukážeme, jak Scrum definuje služby k ostatním rolím v Scrum týmu a organizaci.

ScrumMaster pomáhá vlastníkovi produktu tím, že (Schwaber a Sutherland, 2011):

- hledá techniky pro efektivnější správu produktového backlogu,
- interpretuje cíle, položky a produktovou vizi vývojovému týmu,
- učí vývojový tým jak vytvářet stručný a srozumitelný produktový backlog,
- rozumí a učí jak vytvářet dlouhodobou produktovou vizi v empirickém prostředí,
- rozumí a učí jak aplikovat agilní vývojové principy,
- na schůzkách bývá v pozici moderátora.

ScrumMaster pomáhá vývojovému týmu tím, že (Schwaber a Sutherland, 2011):

- vede vývojový tým směrem k sebe-organizaci a multifunkčnosti,
- učí a vede, jak má vývojový tým vytvářet kvalitní produkty,
- odstraňuje překážky bránicí vývojovému týmu v kvalitní práci,

- koučuje tým k správnému pochopení Scrumu a jeho bezproblémové přijetí.

ScrumMaster pomáhá organizaci tím, že:

- plánuje přijetí Scrum v organizaci,
- školí organizaci (zaměstnance a ostatním zúčastněné složky) osvojit a pochopit Scrum a empirický vývoj produktu,
- nalézá změny pro vyšší efektivitu (Schwaber a Sutherland, 2011).

Jak lze vidět na výčtu činností Scrum Mastera. Scrum master je klíčovou osobou fungování metodiky Scrum ve firmě. Výběr Scrum Mastera je velmi důležitý milník v nasazení Scrumu a jeho úplné použití (Cohn, 2006). Mike Cohn[9] (2006) se zabývá otázkou: kdo by měl vybírat ScrumMastera. Podle něj by to měli být většinou sami členové vývojového týmu. Avšak výběr ScrumMastera musí být ponechaný na alespoň trochu zkušeném týmu se Scrumem. Tedy ne na týmu, který se teprve učí se Scrumem pracovat, jelikož laxní přístup týmu ke Scrumu může vést k výběru laxního ScrumMastera, což může mít za následek selhání Scrumu ve firmě. Proto Mike Cohn, navrhuje sekundární řešení pro výběr ScrumMaster, což je ponechat výběr na odpovídajícím manažerovi, který odpovídá za vývoj. Ve své knize Mike Cohn dále popisuje charakteristiku Scrum Mastera jako skromného člověka, s přirozenou autoritou, který je kooperativní, zkušený a vzdělaný (Cohn, 2009).

Podle Knesla (2009), zajišťuje také chod kanceláře, řeší spory uvnitř týmu a chrání před rušením z vnějšího prostředí. Měl by samozřejmě znát agilní principy a snažit se je aplikovat a učit ostatní členy Scrum týmu.

ScrumMasterem bývá často projektový manažer, vedoucí technického týmu, nebo někdo s přirozenou autoritou. I když Scrum Master nemá takové kompetence jako například projektový manažer. Nemůže direktivně nic nařizovat, nebo dokonce nikoho vyhodit (Cohn, 2006). Cohn (2012l) dále uvádí, že ScrumMaster je projektový manažer 21. století, tedy manažer zaměřený především na proces a na úspěch projektu. Není zodpovědný za náklady, lidské zdroje a další povinnosti v tradičním duchu projektového managamentu. Většinu těchto povinností má vlastník produktu (náklady, rozsah, plán projektu). Zbylou odpovědnost má celý Scrum tým (Cohn, 2012l).

[9] Mike Cohn je zakladatelem Software Mountain Goat, poradenské společnosti, která se specializuje na pomoc společnostem přijmout a zlepšit jejich využití agilních postupů a technik. Je autorem User Stories Applied for Agile Software Development (uživatelských příběhů aplikované pro rozvoj), Agile Estimating and Planning, and Succeeding with Agile (Agilní odhad a plánování, být úspěšný s agilními metodikami). Mike Cohn je spoluzakladatelem Agile Alliance a člen Aliance Scrum (Cohn, 2006).

c) Vývojový tým

Vývojový tým se skládá z expertů a specialistů, kteří vyvíjejí systém ve sprintu. Scrum tým je sebe-organizující a multifunkční (Cohn, 2012m). Sebe-organizující znamená, že nemá žádného definovaného vedoucího. Sám rozhoduje, jak provede určitou práci. Multifunkčnost týmu znamená, že tým nepotřebuje žádného externistu, na jehož práci musí čekat. Tým má všechny náležité schopnosti pro vykonání práce. Scrum tým neobsahuje tradiční role, jako je programátor, designer, tester a podobně. Každý má stejný status. Tým samozřejmě obsahuje specialisty na určitou problematiku (viz podmínka multifunkčnost týmu), ať je to analýza, návrh designu a databází, testovaní, dokumentace či samotné programování. Jde spíš o filosofii, že nikdo není zodpovědný jen za svůj úsek působnosti, ale za celý projekt. Metodika Scrum pomáhá tým stmelovat, vytvářet přátelské a kreativní klima (Cohn, 2012j).

Charakteristika vývojového týmu (Schwaber a Sutherland, 2011).

- Sebe-organizující. Vývojový tým sám určuje, jak bude vykonávat práci definovanou v produktovém backlogu.
- Multifunkční. Tým má všechny náležité schopnosti pro vytvoření přírůstku produktu.
- Každý člen týmu je nazýván vývojář, bez ohledu na funkci nebo práci, kterou vykonává.
- Zodpovědnost za práci má celý tým. I když každý člen týmu může mít jinou specializaci.
- Vývojový tým neobsahuje další podymy,věnující se konkrétním oblastem (testování, analýza apod.).

Velikost týmu by měla být mezi třemi až devíti členy (Schwaber a Sutherland, 2011). V literatuře se často objevuje doporučený počet členů i mezi pěti a devíti (Cohn, 2012j). Mike Cohn definuje počet členů také bonmotem, že velikost týmu by měla být definovaná tím, že vývojový tým nasytí dvěma pizzami. Dále toto vtipné tvrzení doplňuje seriózní statistikou, ve které se sledovalo na skoro 500 projektech efektivita různě početných týmu. Jako neefektivnější tým se ukázali týmy s pěti až sedmi členy (Cohn, 2009). Tříčlenný tým totiž nemůže maximálně využít iterace mezi sebou a nemusí mít všechny schopnosti k dokončení sprintu dle produktového backlogu. Naproti tomu devíti a více členný tým si žádá až příliš mnoho koordinace a nelze u nich použít empirický proces (Schwaber a Sutherland, 2011).

Mnoho týmů, které jsou zvyklé na rigorózní metodiky nebo tradiční vývoj softwaru si nemusí rychle zvyknout, že je nikdo neřídí a neříká jim, co mají dělat. Pravidla Scrumu však

tým učí přebírat zodpovědnost a orientovat se na výsledek sprintu respektive na konečný výsledek celkového vývoje softwaru (Schwaber, 2004).

2.6.5 Činnosti

Scrum se skládá z časově ohraničených (time-boxed) činností, které musí být řádně zkontrolovány. Žádná z činností nesmí být vynechána (Schwaber a Sutherland, 2011).

Celý životní cyklus vývoje softwaru začíná Sprint Planning Meetingem (plánovací schůzkou), ve které se definují položky, které jsou zapsány do Product Backlogu (product backlog). Dále je vytvořen Sprint backlog (těmito a dalšími artefakty se budeme zabývat v další kapitole 2.7.6). Následuje samotný sprint s Daily Scrums (každodenní schůzky). Na konci Sprintu je Sprint Review (vyhodnocení sprintu) a Sprint Retrospective (retrospektivy sprintu). Po té se celý cyklus opakuje (Cohn, 2012m).

Na dalších stranách se budeme jednotlivým činnostem věnovat podrobněji.

a) Sprint

Sprint by měl trvat 2 – 4 týdny, rozhodně ne více než měsíc. Všechny sprinty musí mít stejnou dobu trvání a sprinty na sebe navazují. Během každého spritnu je vytvořen fungující prototyp.

Každý sprint se skládá z: Sprint Planning Meeting (plánovací schůzky), Daily Scrums (denní schůzek), vývojových prací, Sprint Review (vyhodnocení sprintu) a Sprint Retrospective (retrospektivy sprintu). Během sprintu se neprovádí změna složení vývojového týmu, nesnižuje se kvalita cíle sprintu a neprovádějí se žádné změny, které tento cíl sprintu mohou ovlivnit. Vlastník produktu a vývojový tým však může upřesnit rozsah sprintu (Schwaber a Sutherland, 2011).

Maximální délka sprintu – jeden měsíc, zajišťuje snadnou kontrolu a adaptaci na změny v definici výsledného produktu. To snižuje náklady a rizika s tím spojené (Schwaber a Sutherland, 2011).

Na začátku každého sprintu tým definuje počet vlastností systému, které zadají do produktového backlog. Během tohoto sprintu jsou vlastnosti programovány, testovány a začleněny do systému. Na konci sprintu je provedeno hodnocení a demonstrace softwaru zákazníkovi, který poskytuje zpětnou vazbu pro případnou úpravu (Cohn, 2012m).

Vlastník produktu může sprint zrušit před jeho regulérním ukončením. Důvod může být interní změny (např. firemní strategie), externí změny (např. situace na trhu) nebo změny technologických podmínek. Díky délce sprintu je předčasné ukončení jen výjimkou. Předčasné ukončení spritu bývá spojeno s nárůstem zdrojů, musí se totiž přeplánovat další

sprint, vyhodnotit položky z produktového backlogu a rozhodnout, zda je nově vzniklá položka akceptovatelná nebo se znovu začlení do produktového backlogu (Schwaber a Sutherland, 2011).

b) Plánovací schůzka

Plánovací schůzka slouží k naplánování prací, které mají být během sprintu udělány a na kterých se podílí celý Scrum tým. Je časově ohraničená s poměrem: pro jednoměsíční sprint osm hodin, tzn. pro dvoutýdenní sprint čtyři hodiny. Z plánovací schůzky vychází artefakt Sprint backlog, který obsahuje cíl sprintu (Deemer et al., 2012).

Schůzka se skládá ze dvou stejně časově dlouhých částí (Deemer et al., 2012). V první části se řeší, jaký přírůstek přibude na produktu na konci sprintu. A v druhé části, co se musí tedy udělat a aby se dosáhlo onoho přírůstku (Schwaber a Sutherland, 2011).

Na první částí schůzky vlastník produktu prezentuje seřazené položky produktového backlogu vývojovému týmu, který odhaduje počet položek a samotné položky – funkčnosti systému, které budou během sprintu implementovány. Odhadování položek může provést pouze vývojový tým, který má k tomu dané kompetence. Vstupními informacemi pro toto rozhodování jsou produktový backlog, poslední přírůstek na produktu, lidské zdroje pro další sprint a výkon vývojového týmu v přecházejícím sprintu. Po odhadnutí položek následuje definování celým Srum týmem cíl sprintu (Sprint Goal), který udává, čeho má být dosaženo na konci sprintu. Množina položek z produktového backlogu, které mají být vykonané v určitém sprintu, jsou zapsány do sprint backlogu. K cíli sprintu se vývojový tým dostane implementací funkcionality a technologie. Pokud se neodhadne složitost prací, jedná tým s vlastníkem produktu o rozsahu sprint backlogu (Schwaber a Sutherland, 2011).

Ve druhé část schůzky, po definování obsahu sprintu začne vývojový tým řešit, jak dosáhnout cíle sprintu. Vlastník produktu k tomuto procesu pomáhá tím, že objasňuje položky z produktového backlogu. Vývojový tým si může přizvat na tuto schůzku i externí konzultanty na použité technologie nebo na oblast pro kterou je produkt vyvíjen. Základem této schůzky je odhadnutí času pro práci v daném sprintu. Ačkoliv je tento odhad nesmírně těžký, je velmi důležitý. Na konci schůzky vývojový tým musí vlastník produktu a Sctum Masterovi vysvětlit, jak chtějí vytvořit přírůstek na produktu a dosáhnout tak cíle sprintu.

c) Denní Scrum schůzka

Každý den ve stejný čas a na stejném místě probíhá patnácti minutová schůzka (Daily Scrum). Měla by se konat ráno a slouží pro synchronizaci aktivit a tvorby plánu na další den.

Na schůzce je vytvářen odhad množství práce do další schůzky a zároveň se kontroluje, zda naplánovaná práce ze schůzky minulé, byla provedena (Deemer et al. 2012).

Hlavní otázky této denní schůzky dle Schwabera a Sutherlanda (2011) jsou:

- „Co bylo dokončeno od doby poslední schůzky? (včera)
- Co bude dokončeno do termínu další schůzky? (dnes)
- Jaké překážky nám stojí v cestě?“ (Schwaber a Sutherland, 2011, s. 9)

Díky odpovědím na tyto otázky vývojový tým vyhodnocuje postup práce k cíli sprintu, tedy zda dokáže splnit položky ze sprint backlogu. Zároveň odpovědi každého člena týmu na výše uvedené otázky fungují jako skvělá motivace (Cohn, 2012f). Pokud totiž programátor na jedné schůzce řekne, že dnes bude implementovat nějaký modul do systému, je velká naděje, že na příští schůzce bude modul opravdu naimplementován, protože programátor se bude snažit vyhnout ostudě před celým týmem, že úkol, který si sám naplánoval, nesplnil. Na druhou stranu, pokud v jeho výkonu nastal určitý problém, bude se řešit prakticky okamžitě, což je samozřejmě velmi efektivní.

Po schůzce tým přeplánovává zbývající práci ve sprintu. Je důležité zmínit, že na schůzce se neřeší problémy, ty řeší vývojové skupiny až po ukončení schůzky.

Organizátorem schůzky je ScrumMaster. Zajišťuje konání schůzky a hlídá stanovených patnáct minut. Zodpovídá za to, že schůzky se zúčastní jen členové vývojového týmu.

Díky těmto denním kontrolám se snižuje pravděpodobnost chyb a zvyšuje se nalezení problémů a potenciálních rizik. Zlepšuje se komunikace a informovanost členů v týmu o projektu. Zajišťuje kontrolu a adaptaci na změny.

Zajímavé pravidlo těchto schůzek je rozdělení zúčastněných. A to na dvě skupiny, pigs (prasata) a chicken (kuřata). Ve skupině prasat jsou všichni členové Scrum týmu tedy vlastník produktu, ScrumMaster a samotný vývojový tým (Knesl, 2009). Ti mají zároveň povinnou účast na všech schůzkách a mohou bez zábran vstupovat do debaty. Ve skupině kuřat jsou lidé, kteří jsou nepřímo zapojeni do projektu. Jako například obchodník, vývojář z jiného projektu, management firmy atd. Ti samozřejmě nemají povinnou účast, ale co je důležité zmínit, chicken nesmí do schůzky zasahovat – mohou pouze přihlížet (Cohn, 2012f).

d) Vyhodnocení sprintu

Na konci celého sprintu probíhá vyhodnocení sprintu. Náplní této schůzky je zjišťování přírůstku na produktu. Rozhoduje se jak postupovat dále na projektu, a co je nejdůležitější, zákazníkovi se prezentuje meziprodukt. Může také proběhnout úprava produktového

backlogu. Tato schůzka by měla trvat mezi dvěma až čtyřmi hodinami, v závislosti na délce sprintu, (Schwaber a Sutherland, 2011). Vyhodnocení sprintu se skládá z těchto částí:

- vlastník produktu prověřuje, jaké práce byly dokončeny,
- vývojový tým projednává problémy, ale i úspěchy při vývoji,
- vývojový tým prezentuje výsledky sprintu,
- vlastník produktu informuje o aktuálním stavu produktového backlogu a odhaduje datům dokončení,
- Scrum tým navrhuje další kroky, které jsou vstupními parametry pro následující plánovací schůzku.

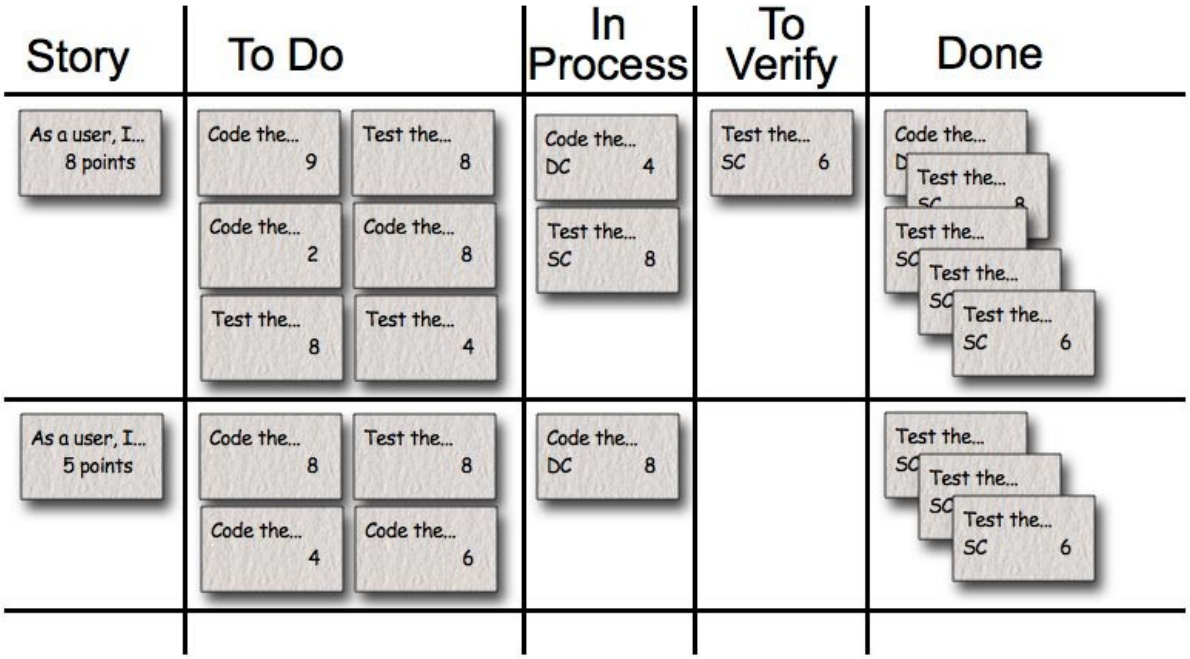
Po této schůzce jsou odškrtnuty hotové položky z produktového backlogu, případně doplněny nové a jsou navrhnuty položky pro další sprint (Schwaber a Sutherland, 2011).

Této schůzky se obvykle účastní vlastník produktu, vývojový tým, ScrumMaster, management, zákazníci, a lidé z jiných projektů (Cohn, 2012h).

Jeff Sutherland na svých stránkách popisuje zajímavou metriku měření rychlosti práce. Píše zde, že tým by měl počítat svoji rychlost v bodech, ne v hodinách. Domnívá se totiž, že lidé neumí počítat s časem a hodnocení v bodech je efektivnější (Schwaber a Sutherland, 2007). Důvody vysvětluje na svých stránkách (Sutherland, 2012).

Jak jsme, již uvedli, Scrum neříká, jaké nástroje použit. Pro řízení projektu a verifikaci, ale Mike Cohn na svých stránkách uvádí jednu z možností, jak řídit projekt pomocí nástroje Scrumboardu (Cohn, 2012i). Scrumboard je vlastně tabulka, která může být reprezentována v nějakém softwarovém nástroji, může být zakreslena na tabuli, nebo formou lístku přišpendlena na tabuli. V Scrumboardu, který má formu tabulky (viz obrázek 2.7), by měl obsahovat sloupce:

- Story – obsahující uživatelské příběhy,
- To Do – obsahující úkoly, které se mají vykonat,
- In Process – úkoly na kterých se aktuálně pracuje,
- To Verify – úkoly na kontrolu,
- Done – zaznamenávající hotové úkoly.



Obr. 2.7 Návrh Scrumboardu (Cohn, 2012i)

e) Retrospektiva sprintu

Nezáleží jak je Scrum tým dobrý, vždy lze něco vylepšit. Mezi vyhodnocování sprintu a plánovací schůzkou je proto Sprint Retrospective (retrospektiva sprintu). Ta slouží k návrhům na zlepšení pro další sprint. Trvá maximálně tři hodiny v závislosti na délce sprintu (Schwaber a Sutherland, 2011).

V této schůzce se kontroluje průběh sprintu s ohledem na lidi, vztahy, procesy a nástroje. Vyzdvihnou se fungující aspekty a navrhnou se věci na zlepšení. Cílem schůzky je zefektivnit další sprint. Výsledkem retrospektivy je seznam věcí na zlepšení pro další sprint. Díky toho se tým učí, zdokonaluje, nestagnuje a zvyšuje svoji efektivitu práce (Schwaber a Sutherland, 2011). Motivovat k těmto zlepšení má za úkol ScrumMaster.

Retrospektiva odhalí mnoho nepříjemných pravd. Každý člověk nerad odkrývá své nedostatky a selhání, ale upřímnost a přiznání chyb vede ke zlepšení procesu a to by si měli členové celého týmu uvědomovat (Schwaber a Sutherland, 2007).

f) Plánování pokerem

V poslední kapitole zabývající se činnostmi podle metodiky Scrum si popíšeme techniku pro ohodnocování položek v produktovém backlogu.

Protože, plánování kolik položek z backlogu se v daném sprintu vykoná a obecně plánování je v procesu vývoje softwaru nesmírně důležité. Scrum nabízí pro časové ohodnocení techniku Planning Poker (plánování pokerem).

Plánování pokerem se používá pro časové ohodnocení položek v produktovém backlog. A to v případě, když se tým nemůže shodnout na složitosti problému a času pro jeho vykonání. Je to tedy technika založená na shodě odhadu (Cohn, 2012e).

Nejdříve se daný problém – úkol – položka z backlogu projedná, aby jí každý rozuměl a pochopil v celé šířce problému. Každý člen pak drží v ruce plánovací karty s hodnoty 0, 1, 2, 3, 5, 8, 13, 20, 40 a 100, které symbolizují čísla uživatelských příběhů, složitost, délku nebo cenu úkolu. Každý člen týmu ohodnotí úkol (položku z backlogu) pomocí těchto karet. Pokud neví, může ohodnotit úkol speciální kartou s otazníkem. Pokud potřebuje čas si složitost problému rozmyslet, ohodnotí úkol další speciální kartou s obrázkem kávy. Všichni odhalují své karty současně, po té se vezme nejčastější hodnota – modus nebo aritmetický průměr hodnot (Baarová, 2012). Pokud se hodnoty značně liší, znovu se diskutuje nad daným problémem a ti, kteří vyložili karty s největší odchylkou, tedy s nejvyšším a nejnižším obodováním, vysvětlí své důvody. Po té se hra opakuje. Pokud opakovaně nedojde ke shodě, může být rozhodnuto o odložení časového ohodnocení do té doby, než se získá více informací o daném problému (Cohn, 2012e).

Tyto hrací karty jsou dostupné v různých obchodech zabývající se metodikou Scrum nebo se plánovací poker dá provádět online přes internet (Mountain, nedatováno).

Plánování pokerem, tedy časové ohodnocení se provádí při psaní produktového backlogu. Ale jelikož požadavky na projekt se vyvíjejí, poker se může hrát i na denních schůzkách. Studie dokázaly, že díky dialogu a více pohledů na problém je tato technika mnohem přesnější v odhadování časové složitosti problému než kterákoliv jiná. Také mnoho softwarových týmů tuto techniku doporučuje, jelikož i jim se v praxi už mockrát osvědčila (Cohn, 2012e).

2.6.6 Artefakty

Artefakty slouží pro kontrolu, adaptaci a poskytnutí transparentnosti. Scrum definuje tři artefakty: Product backlog (produktový backlog), Sprint backlogu (sprint backlog) a Burndown charts (burndown grafy). V oficiálním průvodci metodikou Scrum od autorů Schwaber a Sutherland (2011) je publikován jako další artefakt i takzvaný Increment (přírůstek) a proto často bývá v literatuře uveden jako regulérní čtvrtý artefakt. Po prostudování další literatury jsme však došli k závěru, že autoři berou přírůstek jen jako dodatek k backlogu), proto jej budeme brát tak i my.

a) Produktový backlogu

„Produktový backlog je priorizovaný seznam všeho, co může být potřeba v produktu a je jediným zdrojem požadavků na jakoukoliv změnu v produktu. Vlastník produktu je zodpovědný za obsah, dostupnost a priorizaci backlogu “ (Schwaber a Sutherland, 2011, s. 11).

Produktový backlogu je dynamický, může se upravovat během celého vývoje softwaru. Reaguje na konkurenci, technologie a nové požadavky. Dokud existuje produkt, existuje produktový backlogu (Schwaber a Sutherland, 2011).

Produktový backlog je tedy seznam všech vlastností, funkcí, požadavků, rozšíření a změn, které mají být provedeny. Každá položka v produktovém backlogu má svůj popis, prioritu a odhad. Seznam položek je seřazen podle přínosu, míry rizikovosti, priority a nezbytnosti (Deemer et al., 2012). Položky s největšími prioritami jsou podrobněji a jasněji popsány, jejich odhady jsou tak přesněji stanoveny. Naopak položkám s menšími prioritami je přidělena menší pozornost. Každá položka musí být rozpracována tak, aby ji šlo dokončit během jednoho sprintu (Schwaber a Sutherland, 2011). Zjednodušeně by se dalo říct, že produktový backlog je seznam nevyřízených úkolů (Cohn, 2012b).

Produktový backlog je živý dokument, stále se rozrůstá. „Požadavky na změny nikdy nepřestanou“ (Schwaber a Sutherland, 2011, s. 11). Stejně tak se mění i odhady pracnosti, zpřesňují se detaily a uspořádání položek. Na odhadech spolupracuje vlastník produktu a vývojový tým

Produktový backlog se typicky skládá z (Cohn, 2012c):

- vlastností,
- chyb,
- technické práce,
- získávání znalostí.

Jeden produkt má jeden produktový backlogu. I když na produktu pracuje několik Scrum týmů, je jim k dispozici pouze jeden produktový backlogu (Schwaber a Sutherland, 2011).

Díky produktového backlogu se dá sledovat pokrok k cílovému produktu. Lze přesně vidět zbývající práce, nástrojem k odhadnutí dalšího pokroku mohou být grafy zachycující trendy. Z vyhodnocení jednotlivých sprintů lze měřit produktivita týmu.

Nejpopulárnější a nejúspěšnější cestou jak vytvořit produktový backlog, je s pomocí použití User Stories (uživatelských příběhů), které popisují funkcionality z pohledu uživatele nebo zákazníka.

Uživatelské příběhy jsou krátké příběhy, popisující vlastnosti systému stylem, kdo, co a jak chce udělat (Cohn, 2012g). Forma příběhu by měla vypadat takto:

„As a <type of user>, I want <some goal> so that <some reason>“(Cohn, 2012g).

Často jsou psány na malé kartičky nebo lístky uspořádané na tabuli nebo na stole. O těchto příbězích se pak dále diskutuje. Tak jako funkce – požadavky i tyto uživatelské příběhy se dají dále dělit. Například: Jako uživatel, chci zakoupit zboží. Tento uživatelský příběh dále můžeme rozdělit na: jako uživatel chci seřadit zboží, jako uživatel chci porovnat zboží a tak dále. Tyto příběhy mohou být rozděleny klidně na stovky „podpříběhů“. Obecnější uživatelské příběhy se nazývají Epos (báseň). Uživatelské příběhy se dále dělí pro jejich snadnější pochopení a popsání, ale také že po jejich rozdělení na menší příběhy se snadněji vejdou (implementují) do jednotlivých iterací – sprintů (Cohn, 2012g).

Pokud tedy chceme určitý příběh popsat detailněji, můžeme jej rozdělit. Další možnost je podle Cohna (2012g) začlenit do příběhu podmínku, která nám zaručí potvrzení, že uživatelský příběh je kompletní. Například již zmíněný uživatelský příběh: „jako uživatel chci porovnat zboží“. Můžeme přidat podmínky: „ujisti se, zda zboží je ze stejné sekce (např. notebooky)“, „ujisti se, že jsou vybrány, alespoň dva druhy zboží“.

Psát tyto uživatelské příběhy může kdokoliv, avšak zodpovědnost za to, že uživatelský příběh je z produktového backlogu, má vlastník produktu. Ze samé podstaty agilního vývoje by měl každý člen týmu napsat nějaký uživatelský příběh (Cohn, 2012g).

Protože, příběhy jsou vlastně požadavky na systém. A ty se mohou měnit a přidávat v kterékoliv části životního cyklu projektu. I uživatelské příběhy se mohou dopisovat kdykoliv (Cohn, 2012g).

Je důležité si uvědomit, že uživatelské příběhy nenahrazují produktový backlog. Produktový backlog je v tradiční terminologii rigorózních metodik vlastně dokument požadavků. A ačkoliv psaní uživatelských příběhů se osvědčilo jako nejlepší metoda aplikování položek z produktového backlogu, jsou uživatelské příběhy neúplné bez diskuze o nich. To je hlavní přínos uživatelských příběhů. Psaní uživatelských příběhu ukazuje návod jak uchopit požadavek klienta na systém a jak jej aplikovat (Cohn, 2004).

b) sprint backlog

„Sprint backlog je množina úkolů vybraných z produktového backlogu včetně plánu dodání produktového přírůstku a splnění cíle sprintu. Sprint backlog je prognóza vývojového týmu jaká funkcionalita bude dodána v následujícím přírůstku a kolik práce bude potřebné k dodání této funkcionality“ (Schwaber a Sutherland, 2011, s. 13).

Tak jako produktový backlog i sprint backlog se během sprintu mění. Během sprintu je neustále aktualizován, přidává se do něj další práce a upravují se odhady práce, která ještě nebyla provedena. Upravovat sprint backlog může pouze vývojový tým. Je velmi detailní, jasný, transparentní a ukazuje aktuální obsah zbývající práce do konce sprintu (Schwaber a Sutherland, 2011).

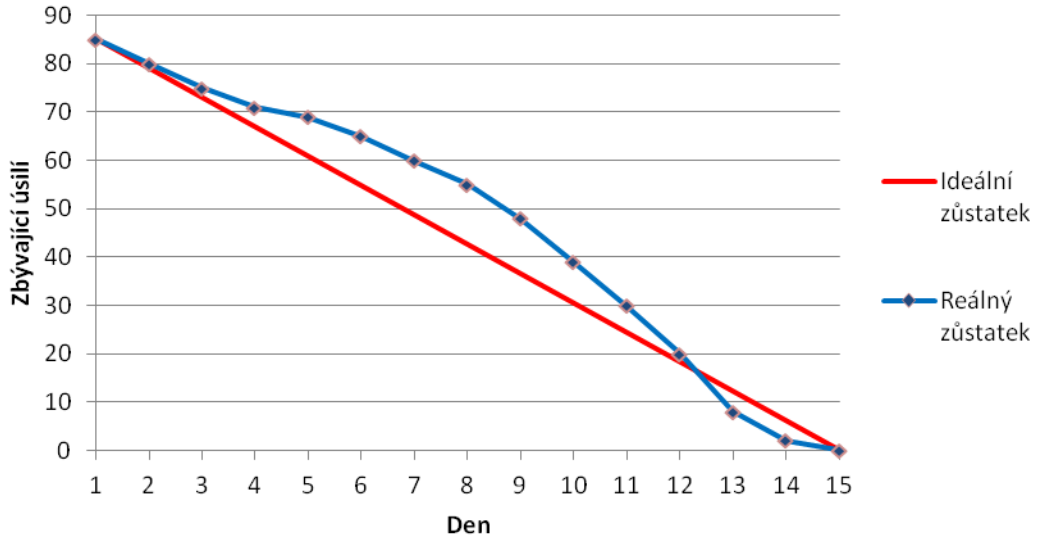
Je důležité, aby bylo vybráno optimální množství práce, které lze reálně provést. Často bývá sprint backlog zapsán pomocí tabulky, ale není to pravidlo (Cohn, 2012k).

Soubor všech dokončených položek ze sprint backlogu se označuje jako Increment (přírůstek). Na konci sprintu musí být tento seznam – přírůstek, hotový. Aby bylo možné jej prohlásit za hotový, musí Scrum tým na začátku jasně a přesně definovat, kdy je možné brát přírůstek za hotový a označit jej za „Done“ (hotovo) (Schwaber a Sutherland, 2011).

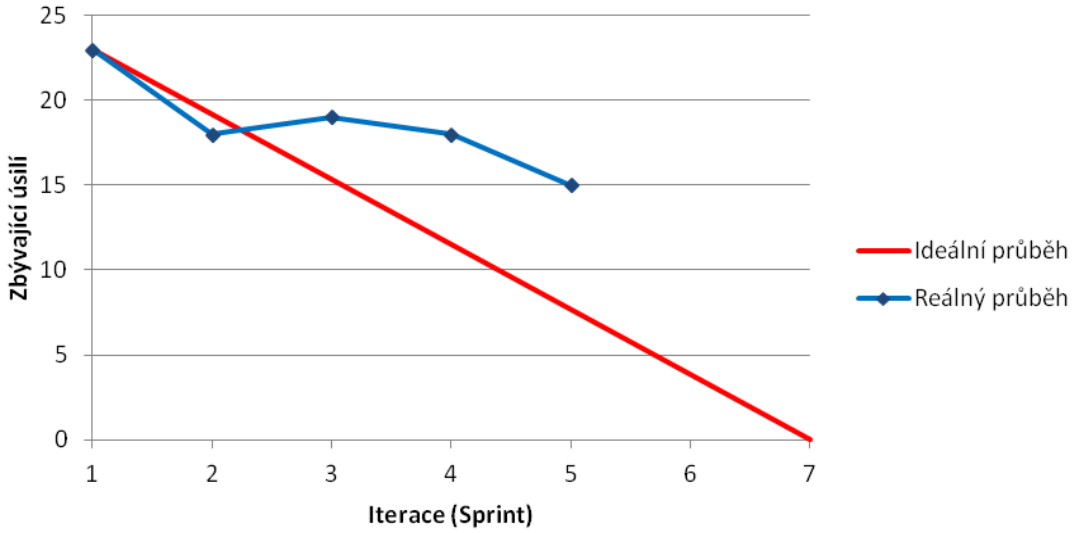
Množství práce, která zbývá do konce sprintu, se sleduje na denní schůzce a vyhodnocuje pomocí grafu.

c) Graf

Posledním artefaktem, který Scrum definuje, jsou burndown grafy, konkrétně Sprint burndown chart a burndown chart uvolnění, tyto grafy ukazují zbývající množství práce v sprintu nebo do finálního předání. Tyto grafy jsou velmi účinné, protože účastnící mají okamžitý přehled o pokroku práce (Cohn, 2012m). Pomocí burndown grafu uvolnění tým dokáže, na konci každého sprintu snadno určit, v jaké fázi procesu jsou, respektive porovnat s plánovaným postupem (Knesl, 2009). Na další straně jsou příklady grafů. Na grafu 2.1 je příklad sprint burndown grafu zaznamenávající reálný průběh práce vůči ideálnímu průběhu v rámci patnácti denního sprintu. Na grafu 2.2 je příklad burndown grafu uvolnění modelující průběh práce po jednotlivých sprintech – iterací proti zbývající úsilí (práce), úsilí může být počítáno také ve dnech.



Graf 2.1 Sprint burndown graf, inspirovaný (International Scrum Institute, 2013)



Graf 2.2 Burndown graf uvolnění, inspirovaný (Cohn, 2012n)

Na vertikální ose je počet bodů/položek z backlogu. Na horizontální ose pak jednotlivé sprinty. Samozřejmě existují i alternativní zobrazení tohoto grafu, ale osy zůstavají stejné (Cohn, 2012a).

Tento graf je nezbytný pro agilní vývoj. Pro dodržení stanoveného termínu. Zároveň slouží jako pomůcka pro řízení projektu. Během vývoje systému, může docházet k nabalování nových požadavků, tento graf dokáže týmu ukázat, že množství nových požadavků může mít za následek zpoždění termínu dodání (Cohn, 2012a).

2.7 Nástroje objektově orientovaného vývoje softwaru

Vedle metodik a modelovacích jazyků pomáhají zefektivňovat vývoj softwaru také různé nástroje zvané CASE (Computer Aided System Engineering) nástroje.

„CASE je počítačový produkt zaměřený na podporu jedné nebo více činností softwarového inženýrství v rámci procesu vývoje informačního systému“ (Brown et al., 1994).

Ze začátku CASE nástroje sloužily pouze k řízení vývoje softwaru (proto název Computer Aided Software Engineering). Při rozvoji těchto nástrojů začaly CASE nástroje sloužit i k plánování, provozu, údržbu a rozšíření systému. Začaly podporovat strategické rozhodování na počátku projektu, operativní činnosti a řízení změn (proto se dnes používá název Computer Aided System Engineering) (Hradecká, 2008).

Dnes CASE nástroje umí například: tvorbu konceptuálního i fyzického datového modelu, objektové modelování, porovnávání modelů, podpora návrhu vícerozměrných schémat, procesní modelování, vytváření datových slovníků, podpora životního cyklu, kontrolní mechanismy, podpora spolupráce při vývoji, generování výsledného kódu, automatické vytváření dokumentace a mnoho dalších funkcí (Hradecká, 2008).

CASE nástroje mohou podporovat, jak strukturované metodiky, tak objektově orientované. Mohou být integrovány do různých systémů a programů podporující tvorbu softwaru. Nástroje CASE se však používají především na rozsáhlejší projekty a ve větších vývojových týmech (Hradecká, 2008).

2.7.1 Nástroje určené pro metodiku Scrum

Jak již jsme uvedli v agilním manifestu, agilní metody, tedy i Scrum dává přednost individualitě a interakci před procesy a nástroji. A Scrum opravdu v zásadě nepotřebuje žádný softwarový nástroj, jeho použití však práci výrazně zjednodušuje, a to v poskytování a sdílení informací. Jelikož jsme si ale uvedli, že Scrum je pouze procesní rámec, může se při vývoji využít další metodiky, nástroje a techniky. Jakou podporu vývojový tým zvolí je čistě na něm. Velmi efektivní může být při větších projektech použít Agile Modeling (Ambler, 2012), a to především modelování Use Cases (případy užití) (Cockburn, 2005). Vývojový tým však nesmí zapomenout na desátý princip z Manifesto for Agile Software Development (Beck et al., 2001a), který doporučuje, dělat jen ty nejdůležitější činnosti tzn. Nemodelovat, pokud to není nezbytně nutné a vytvářené modely jsou k užitku.

Nástrojů podporujících metodiku Scrum a usnadňujících plánování procesu vývoje softwaru je mnoho. Některé jsou placené, některé jsou volně dostupné. V poslední době se

hojně vyskytují především programy na mobilní zařízení (mobily, tablety), které využívají především manažeři softwarových projektů.

Ve většině případů programy pracují ve webovém prostředí a podporují export dat pro prezentaci. Jelikož metodika Scrum je relativně „jednoduchá“ i programy jsou převážně jednoduché, uživatelsky přívětivé a nenabízí příliš mnoho funkcí, které by ve finále spíš jen brzdily projekt a byly by kontraproduktivní (Fianta, 2011).

Po prostudování literatury, která se zmiňuje o metodice Scrum a je uvedena na konci této práce, můžeme pro malou a střední firmu doporučit nějakou webovou aplikaci, pro sdílení ScrumBoardu, produktového backlogu, sprint backlogu a ve které se dají jednoduše vytvářet burndown grafy. Co bychom dále mohli doporučit, je pořízení karet pro plánování pokeru. Pro distribuované týmy je možnost odhadovat čas pomocí plánovaní pokeru online (Mountain Goat Software, nedatováno).

2.8 Unified Modeling Language

„UML (Unified Modeling Language) je univerzální jazyk pro vizuální modelování systémů“ (Arlow a Neustadt, 2007, s. 28), není tedy metodika. UML vzniklo v roce 1997, kdy jej sdružení OMG (Object Management Group – standardizační skupina) přijalo jako standart objektově orientovaného jazyka pro vizuální modelování. UML bylo vytvořeno ve firmě Rational. Hlavní autorství se připisuje trojici Grady Booch, Ivar Jacobson a Rumbaugh, kteří spojili nejlepší existující postupy modelovacích technik a softwarového inženýrství[10]. Do té doby existovalo mnoho modelovacích jazyků, avšak neexistoval žádný standard, který by byl nezávislý na některé metodice a mohl tak sjednotit vizuální modelování. Jediný vážný adept byla metodika Fusion, ta se však neuplatnila. Aktuálně je UML ve verzi 2.0.

UML je navrženo tak, aby všechny nástroje CASE jej mohli implementovat a interpretovat. Zároveň diagramy vytvořené v jazyce UML jsou snadno čitelné i pro lidi. UML neobsahuje žádnou metodiku modelování, jazyk UML poskytuje pouze vizuální syntaxi. Zároveň není vázán se žádnou metodikou nebo fází životního cyklu. Některé metodiky však UML jazyk obsahují v sobě. Jako například UP nebo OPEN.

[10] Některé postupy převzali ze svých vlastních metod a metodik. Konkrétně z OMT (Object Modeling Technique) vytvořenou Boochem a Rumbaughtem a z metodiky Objectory od Ivara Jacobsona (Kanisová, 2004).

Základní výhody jazyka UML jsou podle Arlowa a Neustadta (2007) následující.

1. UML jazyk lze použít v jakékoli fázi vývojového cyklu softwarového procesu.
2. Lze jím modelovat jakoukoliv aplikační doménu.
3. UML je nezávislí na programovém jazyce nebo platformě. (Podporován je, ale hlavně v objektově orientovaných jazycích.)
4. UML je nezávislá na metodice, modelu či schématu procesu vývoje softwaru.
5. Konzistence uvnitř jazyka UML je zajištěna vlastními interními pojmy.

UML pracuje se souborem spolupracujících objektů, na které pohlíží ze dvou úhlů (Arlow a Neustadt, 2007).

- Statická struktura – popisuje typy objektů a souvislost mezi nimi.
- Dynamické chování – popisuje životní cyklus objektů a způsob jejich spolupráce.

Struktura UML2 dle (Arlow a Neustadt, 2007) se dělí na stavební bloky, společné mechanizmy a architekturu. Tuto strukturu si popíšeme v následujících podkapitolách. Pokud nebude uvedeno jinak, hlavním zdrojem nám byla kniha autorské dvojce Arlow a Neustadt, (2007).

2.8.1 Stavební bloky

UML je sestaveno ze tří stavebních bloků. Z předmětů (things), vztahů (relationship) a diagramů (diagrams).

a) Předměty

Předměty jsou samotné prvky modelu.“ Předměty jsou podstatnými jmény modelu UM.“ (Arlow a Neustadt, 2007, s. 35). Předměty se dále dělí na:

- struktury abstrakce, které jsou podstatnými jmény UML, mezi které patří třídy, případ užití, rozhraní, komponenta nebo uzel,
- chování, které vyjadřují slovesa modelu UML jako např. interakce nebo stav,
- seskupení, jsou balíčky sémanticky seskupených prvků modelu UML,
- poznámky, anotace.

b) Vztahy

Vztahy nebo taky relace zachycuje vztah mezi dvěma předměty. Tyto vztahy lze rozdělit do několika typů relací dle vztahu, který symbolizují:

- asociace – spojení mezi objekty,
- závislost – změna v jednom předmětu ovlivňuje druhý předmět,
- agregace – jeden předmět je součástí jiného předmětu,
- kompozice – alternativa agregace s více omezeními,
- ochranná nádoba – další alternativa agregace, používá se především s balíčky,
- zobecnění – jeden prvek je specializací obecného prvku druhého,
- realizace – asociace mezi klasifikátory.

c) Diagramy

Vytvořené předměty a relace mezi nimi se ve většině CASE nástrojů, založených na UML automaticky přidají do vznikajícího modelu. „Model je repositářem, všech předmětů a relací vytvořených k tomu, aby popisovaly požadované chování systému, který se snažíme navrhnout“ (Arlow a Neustadt, 2007, s. 36). Diagramy jsou pohledy do těchto modelů. UML2 definuje třináct diagramů, které dělí do dvou skupin. První skupinou jsou diagramy struktury, modelující statickou strukturu systému (statický model). A druhou skupinou jsou diagramy chování, modelující chování neboli dynamickou strukturu systému (dynamický model). Diagramy rozdělené do skupiny dle UML2:

- diagram struktury,
    - diagram tříd,
    - diagram složené struktury,
    - diagram komponent,
    - diagram nasazení,
    - objektový diagram,
    - diagram balíčku,
- diagram chování,
    - diagram aktivit,
    - diagram interakce,
        - sekvenční diagram,
        - diagram komunikace,
        - stručný diagram interakce,
        - diagram časování,
    - diagram případu užití,
    - stavový diagram.

2.8.2 Společné mechanismy

Společné mechanismy jsou čtyři strategie modelování objektů používány opakovaně napříč v celém jazyku UML. Tyto čtyři obecné mechanismy jsou specifikace, ozdoby, podskupiny a mechanismy rozšiřitelností.

a) Specifikace

Diagramy umožňují zobrazovat model graficky. Textový rozměr v UML zastupuje specifikace, která slovně popisuje sémantiku prvků. Specifikace je jádrem modelu a tvoří sémantický podklad pro vizualizaci diagramy. Model zobrazen pouze pomocí diagramů, bez specifikace je neúplný. Modely mohou být pak proškrtané – prvky jsou ve specifikaci, ale nejsou v žádném diagramu. Nebo neúplné – prvky v modelu chybí úplně. A posledním případem, prvky mohou být nekonzistentní – tedy prvky jsou ve specifikaci a v diagramu jiné.

b) Ozdoby

Každý symbol zobrazující nějaký prvek, lze obohatit ornamentem za účelem zdůraznění či zvýraznění nějakého detailu. Adornments (Ozdoby) bývají někdy překládány jako ornamenty.

c) Podskupiny

UML se dívá na svět různými způsoby, které rozděluje do dvou podskupin. První podskupinou jsou klasifikátory a instance a druhou podskupinou rozhraní a implementací.

„Klasifikátor je abstraktním vyjádřením typu předmětu. Instance je naproti tomu konkrétním výskytem abstraktní představy“ (Arlow a Neustadt, 2007, s. 42). Typy klasifikátorů jsou aktér, třída, komponenta, role klasifikátoru, datový typ, rozhraní, uzel, signál, subsystém, případ užití a dalších dvacet tři klasifikátorů. To, co předmět vykonává, označujeme jako rozhraní, a to jak to vykonává, nezýváme implementací. Předmět tedy má definované rozhraní zásad, co všechno může. Implementace pak je, jak se dané zásady z rozhraní budou s těmito zásadami realizovat. Implementace rozhraní může být pokaždé a v různých případech provedena jinak.

d) Mechanismy rozšiřitelnosti

Aby UML bylo opravdu univerzální a byly uspokojeny potřeby všech uživatelů, jsou do něj začleněny tyto mechanismy rozšiřitelnosti: omezení, stereotypy a označené hodnoty. Omezení je text, specifikující podmínky a pravidla určitého prvku modelu, kde tyto podmínky musí být ohodnoceny jako pravda. Jako standardní omezení UML definuje omezující jazyk OCL (Object Constraint Language).

„Stereotyp zastupuje určitou variantu v daném modelu existujícího prvku, který má sice stejnou podobu (atributy a relace), ale používá se s jiným záměrem“ (Arlow a Neustadt, 2007, s. 43). Stereotypy slouží k vytváření nových prvků založených na již existujících.

Většina prvků v UML mají předefinované vlastnosti. Tyto vlastnosti lze rozšířit pomocí označených hodnot, které jsou zobrazeny jako seznam dvojic ve tvaru značka = hodnota.

Profil UML je soubor omezení, stereotypů a označených hodnot sloužící pro konkrétní problém. Profily UML upravují a přizpůsobují UML tak, aby jejich použití bylo efektivní, konzistentní a jasně definované. Příkladem takového profilu je například UML pro .NET pro modelování aplikace na základě .NET.

2.8.3 Architektura

James Rumbaugh, Ivar Jacobson a Grady Booch ve své příručce k UML definují architekturu jako „Organizační struktura systému, včetně jeho rozkladu do částí, jejich propojení, interakční mechanismy, a řídicí principy, která vzniká v návrhu systému“ (Rumbaugh, Jacobson a Booch, 1999, s. 150). Standard IEEE definuje architekturu systému jako nejvyšší úroveň koncepce systému v jeho vlastním prostředí (Arlow a Neustadt, 2007, s. 46). Způsobů, jak zachytit různé aspekty architektury, je spousta. Jeden z nejznámějších způsobů je pohled 4+1 (označován taky jako architektura 4+1) od Philippa Kruchtena (Kruchten, 1995). Tento způsob pohledu využívá a definuje i UML nebo RUP (IBM, 2007). Pro zachycení všech aspektu architektury, UML tedy definuje čtyři různé pohledy na systém: logický pohled, pohled procesů, pohled implementace a pohled nasazení. Tyto pohledy jsou pak součásti pátého pohledu, pohledu případu užití.

Seznam a popis jednotlivých pohledů dle Kruchtena (1995) a dvojce Arlow a Neustadt (2007).

- Logical View (Logický pohled), znázorňuje logickou strukturu systému a jeho funkcionalitu. Zaměřuje se na chování tříd a objektů.
- Process View (Pohled procesů). Procesně orientovaná varianta logického pohledu. Zaměřuje se na procesy, chování tříd, a dynamiku systému.
- Development View (Pohled implementace). Je pohled organizace statických softwarových komponent. Modeluje soubory a komponenty a závislosti mezi nimi.
- Physical view (Pohled nasazení). Modeluje fyzické nasazení softwaru na hardware.
- Scenarios (Pohled případu užití). Zachycuje požadavky na systém. Z tohoto pohledu se odvíjí a čerpají předešlé pohledy. Je to tedy jádro architektury 4+1.

3 Analýza stávajícího stavu použití metodik ve firmě

Cílem práce je navrhnou a implementovat metodiku dle Scrum do firmy Poski. Abychom toto dokázali, musíme nejdříve analyzovat aktuální stav procesu vývoje softwaru ve firmě, kterou se budeme zabývat v první části této kapitoly.

Metodika Scrum je pouze procesním rámcem a postihuje pouze některé aspekty procesu vývoje SW. Metodika například kdy a jak testovat, nebo tvorbu dokumentace. Tyto praktiky a rozhodnutí nechává na vývojovém týmu, který nejlépe ví, co, kdy a jak má udělat, aby splnil cíl sprintu.

Abychom nalezli odpovědi i na některé dílčí problémy, podíváme se na proces vývoje softwaru ve střední firmě D3Soft, která vyvíjí agilním způsobem. Obsahem druhé části tedy bude již ne tolik podrobná analýza vývojového procesu. Cílem analýzy v D3Soft bude především na porovnání procesů mezi oběma firmami a nalezení inspirací pro pozdější návrh metodiky.

Při popisu procesu vývoje v obou firmách jej zjednodušíme na hranici fází a iterací v nich. Některé procesy při vývoji ve firmách nebudeme dokonce zmiňovat. Některé zobecníme a nazveme je pouze jako výroba. Scrum a tedy ani my například nebudeme řešit, kdy se mají po naprogramování některé funkce testovat, nebo kdy se má do návrhu designu zapojit programátor.

Zdrojem[11] nám v obou firmách byly interní materiály podniku a rozhovory s managementem podniku a zaměstnanci. Rozhovory byly vedeny neformálně a neměly pevně stanovenou strukturu.

3.1 Analýza společnosti Poski.com

Firma Poski se snaží již dva roky nasadit do svého výrobního procesu metodiku Scrum. V této kapitole se budeme snažit o analýzu stávajícího stavu použití této metodiky a zaměříme se na hlavní problémy a příčiny neúspěšného nasazení této metodiky. Nejprve uvedeme základní informace o firmě, její historii, činnost a aktuální ekonomické situaci. Následovat bude její organizační struktura, včetně zachycení struktury diagramem. Dále se budeme zabývat samotným procesem výroby, zahrnující slovní popis procesu, definování rolí, artefaktů a činností. Vývojový proces zachytíme diagramem aktivit a provedeme SWOT analýzu. Porovnáme, které prvky z metodiky Scrum firma používá, které naopak vynechává a proč.

[11] Kromě literatury uvedené v teoretické části nám byli pro zpracování návrhu vzorem také knihy od Page-Jonese (2001), Brittona a Doakea (2005).

V analýze vývojových procesů se budeme zabývat vývojem webových stránek, CMS, e-shopu, CRM, PoksiReal a další vývoj internetových aplikací na zakázku (více o těchto produktech na další straně).

Implementace těchto šablonových produktů a typu softwaru se od sebe samozřejmě liší. Cílem analýzy je však celý vývojový proces softwaru podle metodiky Scrum. V této metodice je implementační činnost obsahem Sprintu a tým implementaci řeší podle svého nejlepšího uvážení a zkušeností. Proto v celé analýze budeme brát vývoj těchto aplikací jednotně označovaný prostě software nebo systém.

3.1.1 Historie a činnost firmy

Firma Poski byla založena v roce 1998 Tomášem Poskerem. V té době se zabývala pouze tvorbou webových stránek. V roce 2003 se Poski spojilo se společností podobného zaměření a vnikla firma PA Holoding, to vedlo k rozšíření nabídky produktů a zkvalitnění služeb. V roce 2004 byla firma přejmenována na Poski.com a rozšířila své zaměření i na web design, web hosting[12], CRM a CMS systémy, internetové aplikace, reklamní kampaně, grafiku, SEO optimalizace webových stránek, e-business, SEM a další služby internetu.

V následující analýze se budeme zabývat procesem tvorby Poski produktů: webové stránky, CMS, e-shopy, CRM, PoksiReal a další vývoj nespecifikovaných internetových aplikací na zakázku. V dalších řádcích si tyto hlavní softwarové produkty společnosti popíšeme[13].

- Webová stránka – klasická webová prezentace, buď vytvořena úplně nová podle požadavků klienta nebo použití šablony z burzy Poski.com vhodná pro méně náročné klienty.
- PoskiQ – CMS je redakční systém určený pro správu textového a grafického obsahu webové prezentace přímo administrátorem bez znalosti programování. Jedná se o originální a kvalitní CMS řešení z dílny Poski.com s možností individuálních úprav přímo na míru konkrétního zájemce. Redakční systém je modulový, umožňuje víceúrovňový přístup a obsahuje ve všech modulech vizuální editor umožňující správu obsahu kterékoliv stránky internetové prezentace.

[12] Pronájem prostoru pro webové stránky na cizím serveru.
[13] Popisy produktů byly poskytnuty společností Poski,com.

- Poski eshop – a elektronické obchod v prostředí internetu. Firma Poski.com nabízí tři základní krabicové řešení[14], které jsou rozděleny podle náročnosti požadavků: Poski eShop Start, Poski eShop Easy, Poski eShop Klasik. Poski také nabízí možnost vývoje e-shopu na míru, označen pod názvem Poski eShop Business.
- Poski CRM – je manažerským řídicím systémem pro malé a střední firmy. Slouží pro správu zákazníků a projektů a je založený na on-line webovém rozhraní. Jedná se o modulový systém s možností konkrétních úprav pro danou organizaci". Pokračování citace: „V současné základní modulaci je Poski CRM vhodný zejména pro obchodní a výrobní firmy střední velikosti, které chtějí mít přehled o svých obchodních kontaktech a komunikaci s nimi. Poski CRM poskytuje možnost mapování průběhu obchodních případů a výrobních procesů a samozřejmě také kompletní fakturaci.
- PoskiReal – je manažerský řídicí systém pro realitní kanceláře založený na on-line webovém rozhraní. Jedná se o modulový systém s možností individuálních úprav pro konkrétní realitní kancelář. Pokrývá potřeby práce makléřů, malých, středních i velkých realitních kanceláří.

Dnes má firma dvacet pět zaměstnanců a v ostravském regionu je jedním z hlavních hráčů na poli internetových aplikací.

Firma Poski nevyvíjí momentálně podle žádné normy, standardu, certifikátu nebo metodiky.

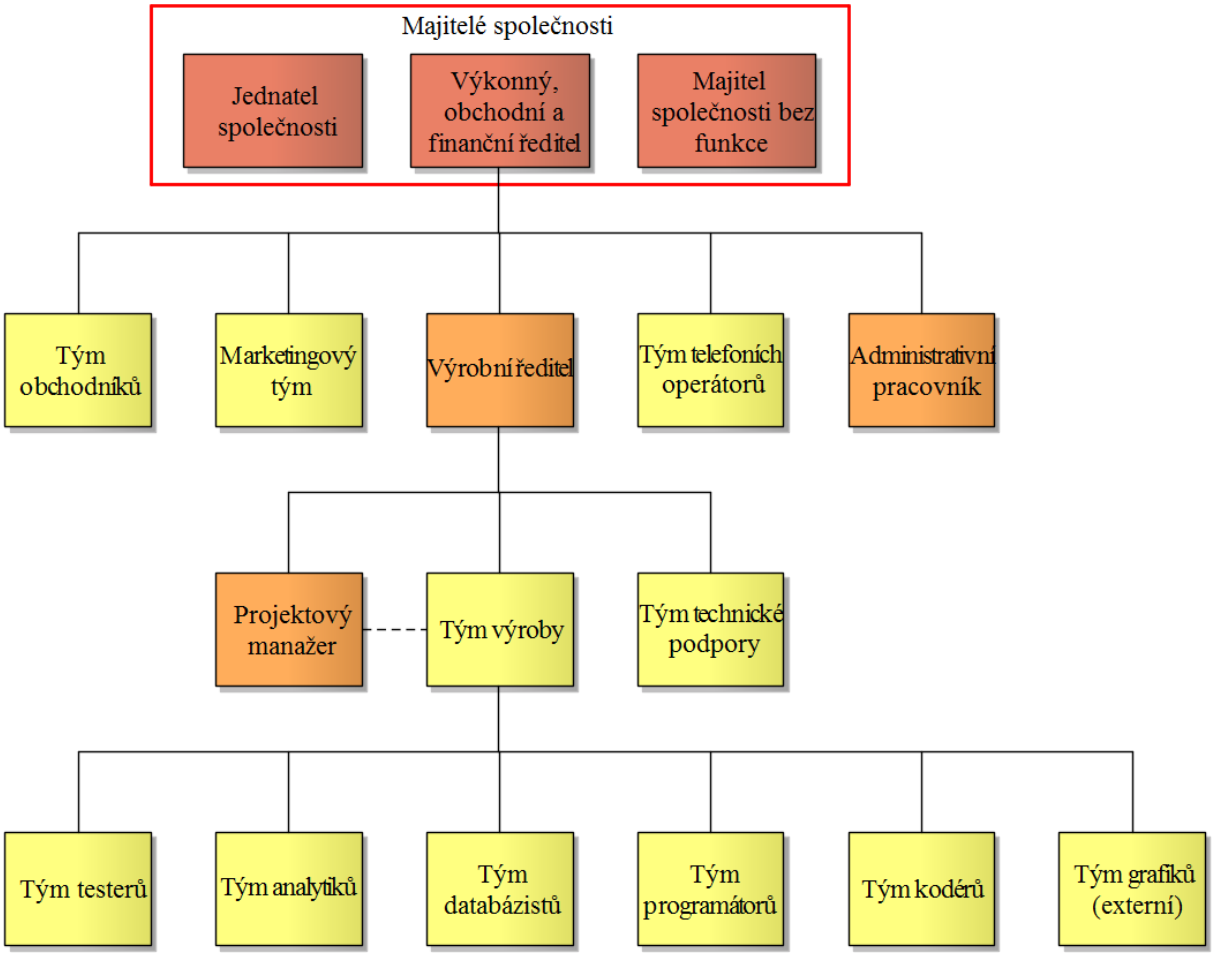
3.1.2 Organizační struktura

Abychom správně pochopili procesy ve firmě Poski, na dalších řádcích si popíšeme organizační strukturu.

Majitelé společnosti jsou tři osoby, z toho dvě mají ve firmě některé z funkcí. Jeden z nich je jednatelem společnosti, druhý se výkonně podílí na běhu společnosti, zastupuje ve firemní hierarchii výkonného, obchodního a zároveň finančního ředitele. Tento vedoucí má pod sebou týmy obchodníků, telefoniích operátorů a marketingový tým. Dále má pod sebou administrativního pracovníka výrobního ředitele. Poslední z nich má nestarosti celou výrobu. Spadá pod něj tým technické podpory a tým výroby včetně projektového manažera. Tým výroby se skládá z testerů, analytiků, specialistů na databáze, programátorů, kodérů

[14] Krabicovým řešením bývá označován software, který se nijak neupravuje, nebo jen s malými úpravami a bývá nabízen prodáván jako již hotové řešení.

a externích grafiků. Nutno uvést, že některé osoby působí ve více rolí, např. tester je i zároveň programátor. Organizační strukturu dokládá obrázek 3.1.



Obr. 3.1 Organizační struktura společnosti Poski.com

3.1.3 Popis nasazení metodiky Scrum

Metodiku Scrum se firma snažila poprvé nasadit v roce 2011 a to na dva projekty. Projekty byly střední velikosti a náročnosti. Podle ředitele výroby byla splněna všechna pravidla a dodrženy všechny role, artefakty a činnosti, které Scrum definuje. Hlavní příčina, proč management od metodiky opustil, podle nich byla:

1. Nevhodnost metodiky Scrum pro jejich projekty. Nemožnost řídit více projektů najednou, protože v časovém rámci sprintu se paralelně pracovalo průměrně na pěti projektech. Projekty se tedy neprováděly sériově za sebou (jeden skončí, druhý začíná), ale více projektů najednou – paralelně. Vývojáři přeskakovali z jednoho projektu k druhému. Navíc projekty se dynamicky pozastavovaly a znovu spouštěly v závislosti čekání

na vyjádření klienta k produktu, nebo pozastavení projektu kvůli finančním problémům klienta. To mělo za následek neefektivní režii řízení projektů.

Další příčiny, které jsme nalezli po rozhovorech s managementem, jsou:

2. Tým nebyl sebe-organizující. Nepřebral plně na sebe zodpovědnost a stále byl řízen a veden někým zvenčí – managementem. Týmy si sami nevolí jak danou práci provést.
3. Tým nebyl multifunkční. V týmu chyběli grafici, kteří byli najímání pouze externě. Kvůli této externí vazbě musel tým čekat na návrh grafiky, což zpomalovalo celý proces vývoje SW. Do problému multifunkčnosti zahrneme také problém, že firma neměla personálně oddělený tým podpory. Někteří členové vývojového týmu se museli věnovat také podpoře a servisu hotových aplikací, přičemž tyto nárazové a neočekávané úkoly jim zabíraly čas. To mělo za následek znesnadnění alokaci lidských zdrojů, zpomalování vývoje SW a zpožďování termínu ukončení sprintu.
4. Špatné odhady časové náročnosti úkolů. Tým nebo respektive jedinci měli obecně problémy odhadnout časovou náročnost svých úkolů. To mělo za následek zkreslení celého časového návrhu projektu. K tomu nežádoucímu efektu přispíval také již zmíněný fakt, migrace některých pracovníků k práci na podpoře a servisu.
5. Vlastník produktu neměl všechny kompetence, které definuje Scrum.
6. Nebyl definován ScrumMaster, který by se čistě věnoval pouze učení a dodržování metodiky Scrum. Ačkoliv byla vyhrazena osoba pro tuto funkci, nebyla dostatečně kompetentní a nesplňovala všechny požadavky na správného ScrumMastera.

3.1.4 Analýza stávajícího stavu

a) Popis procesu vývoje softwaru

Firma Poski vyvíjí software dvěma způsoby v závislosti na velikosti požadovaného systému.

První způsob aplikuje u menších projektů, kde zákazník klade velký důraz na rychlost dodání softwaru a na výslednou cenu. Ve většině případů klient nemá velkou představu o tom, co vlastně chce, jaké funkce si přeje a co by mělo být výsledkem vývoje. Proto jsou využívány šablony a již existující komponenty ze starších projektů. To zapříčiňuje omezení originality a do jisté míry i kvality výsledného produktu. Jde de facto o tradiční vodopádový model. Tedy o model nevyhovující dnešním podmínkám, především v oblasti internetových zakázek. Hlavním problémem je, že firma vytvoří na konci procesu nějaký funkční produkt v definovaném čase, který však nesplňuje klientovi představy. Vývojový tým se proto dostává

na začátek celého procesu vývoje, znovu probíhá konzultace s klientem a software se upravuje a předělává. Tím se celý projekt prodlužuje. Klient může být s prodloužením délky dodání nespokojen a může odmítat zaplatit nadbytečnou práci. Tím firma přichází jak o zisk, tak o uvolnění zdrojů na další projekty. A v jistých případech i nelichotivou referenci. Je tedy nutné provést několik menších iterací, vytvářet prototypy a více zapojit zákazníka do výroby. Je zřejmě, že tento způsob vývoje nemá s metodikou Scrum nic společného. Výhodami využití agilních metodik, jsme se již zabývali napříč celou touto diplomovou prací. Pokud chce tedy firma Poski být i u těchto projektů efektivní, je třeba pohlížet na tyto malé projekty jako na projekty velké a vyvíjet i v tomto případě dle metodiky Scrum. Prvním způsobem se tedy dále nebudeme zabývat, respektive budeme oba způsoby brát jako jeden. A to jak v analýze, tak následné implementační části.

Druhý způsob vývoje se využívá u velkých projektů. Kde je taky samozřejmě kladen důraz na čas a cenu. Ale není již tak velký a klientovi záleží hlavně na kvalitě, nadstandardní nebo nových funkcí a na grafické originalitě.

Seznam artefaktů:

- nabídka,
- smlouva,
- specifikace požadavků,
- testovací protokol,
- dokumentace,
- předávací protokol,
- manuál.

Firma Poski dnes nepoužívá žádné Scrum artefakty jako jsou produkt backlog, sprint backlog ani burndown grafy. Pokusy o tyto artefakty sice byly, ale pouze jednorázově a firma od nich brzo opustila. Produktový backlog dnes částečně nahrazuje specifikace požadavků.

Scrum definuje tři role: ScrumMaster, vlastník produktu, a vývojový tým. Scrum Master a vlastník produktu nejsou ve firmě definovaní. Část kompetencí vlastníka produktu přebírají obchodní zástupce a projektový manažer. Scrum Master v týmu úplně chybí.

Role ve vývojovém týmu Poski jsou:

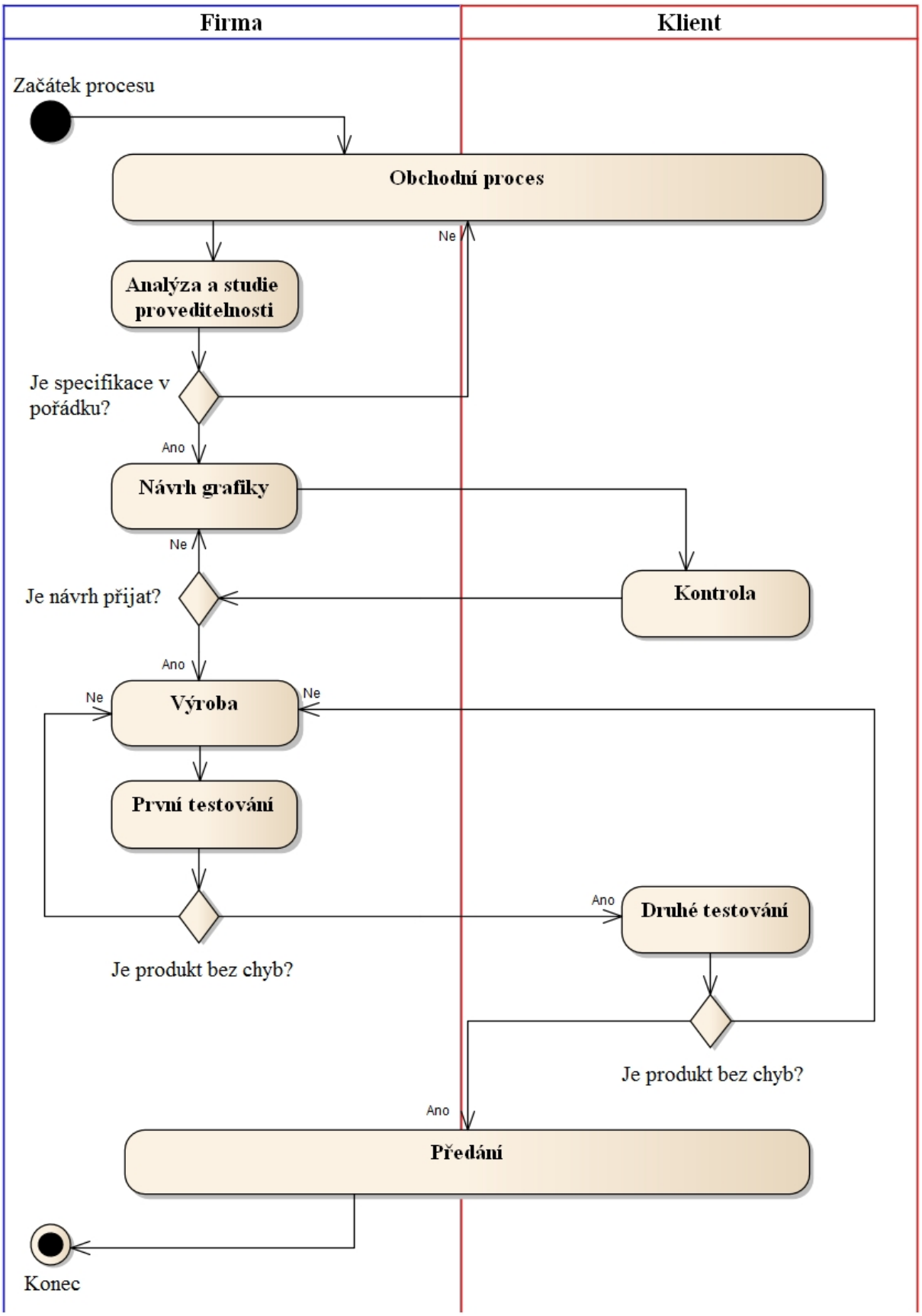
- obchodní zástupce,
- projektový manažer,
- analytik,
- grafik,
- programátor,
- kodér,
- databázista,
- tester.

Celý proces začíná navázáním kontaktu s klientem, a to ať ze strany klienta nebo firmy. Probíhá jednání o projektu, prezentace nabídky společnosti, zmapování požadavků klienta a nakonec dohoda o spolupráci. Všechny tyto činnosti zahrneme do aktivity nazvané obchodní proces, ve kterém figurují obchodní zástupce a klient. Výstupem je specifikace požadavků, která je zároveň vstupním artefaktem pro následnou analýzu a studii proveditelnosti. Na této aktivitě se podílí analytik, projektový manažer, obchodní zástupce a někdy programátor. Tato činnost obsahuje technické a technologické řešení, návrh harmonogramu a především studie proveditelnosti v rámci stanoveného rozpočtu. Pokud je specifikace požadavků a rozpočet na projekt schválen, následuje aktivita návrh. V opačném případě se pokračuje v obchodním procesu. V návrhu se připraví návrh grafiky, případně se implementují základní optimalizované komponenty a moduly. Aktéry jsou grafik a programátor. Následuje kontrola klientem a po jeho odsouhlasení se přechází k samotné výrobě. V opačném případě se návrh upravuje a znovu dochází ke kontrole. V této iteraci se pokračuje až do schválení zákazníkem. Pokud jde o malé grafické úpravy, paralelně se začíná už i s výrobou. V aktivitě výrobě nedochází k žádné iteraci nebo kontrole zákazníkem. Na výrobě se podílí programátor, kodér, a databázista, kteří vytvoří fungující konečný produkt. Ten je nejdříve otestován testery a pak klientem. V případě neúspěšného testování, nalezení chyb nebo nesrovnalostí s požadavky či představy klienta, se program dostává znovu do výroby a upravuje se. Největší problém nastává, když klient nalezne nesrovnalost se svým požadavkem nebo představou v základní programové struktuře a systém se musí předělávat úplně od začátku. Tato smyčka se opakuje až do konečného schválení klientem, po kterém následuje předání, tedy nasazení a instalace fungujícího softwaru. Tím samozřejmě životní cyklus softwaru nekončí, následuje udržování a rozvoj softwaru. Protože těmito fázemi se metodika Scrum nezabývá, nebudeme jej

popisovat ani my. Popisovaný proces zachytíme pomocí UML konkrétně diagramem aktivit, který je na obrázku 3.2. Do diagramu respektive modelu jsme nezahnuli artefakty. V modelu jsme znázornili rozdělení aktivit podle rolí na interní (firma) a externí role (klient). Rozdělení aktivit ve firmě dle interních rolí jsme pro přehlednost nezakreslili.

Společnost Poski průměrně pracuje paralelně na pěti projektech v rámci jednoho měsíce. Další desítky projektů jsou však pozastaveny, kdy se čeká na další krok klienta nebo třetí strany. Důvodem může být například nedostatek zákazníkových finančních zdrojů pro pokračování realizace projektu.

Firma vyvíjí software především pomocí jazyku PHP a technologie .NET, ale také pomocí dalších programovacích jazyků a standardů pro vývoj a prezentaci internetových aplikací. Pří vývoji firma Poski nepoužívá žádné nástroje nebo techniky pro modelování. V závislosti na obtížnosti a robustnosti vyvíjených aplikací je modelování zbytečné.



Obr. 3.2 Diagram aktivit popisující aktivity při vývoji softwaru ve společnosti Poski.com

b) Časové rozložení jednotlivých fází vývoje softwaru

Další analýza obsahuje časové rozložení jednotlivých fází vývoje softwaru. Do tabulky 3.1 jsme zaznamenali, jak dlouho trvá každá fáze, které jsme si nadefinovali v teoretické části. Fáze udržování systému, rozvoj systému a stažení systému budou vynechány, protože délka jejich trvání je proměnlivá a nedá se změřit. Délku každé fáze jsme vyjádřili v procentech vůči celkové délce trvání projektu.

Fáze vývoje softwaru	**Doba trvání v %**
Byznys modelování	0
Specifikace požadavků	15
Analýza	15
Návrh	10
Implementace	40
Testování	10
Nasazení	5
Vyhodnocení projektu	5

Tab. 3.1 Doba trvání fází vývoje SW vzhledem k celkovému času

Z analýzy délky jednotlivých fází životního cyklu softwaru jde vidět (tab. 3.1), že ačkoliv firma Poski nevyvíjí podle žádné metodiky, dá se říct, že samovolně inklinuje k agilnímu přístupu. Zaměřuje se totiž především na implementaci – 40 %. Z povahy projektů je jasné, že byznys modelování nemá při vývoji nijak zásadní úlohu a z agilního pohledu je dobře, že se firma nevěnuje zbytečným činnostem. Z tabulky se dá vypozorovat, že ale specifikaci požadavků a návrhu by mohla firma věnovat více času. Možný problém můžeme také vidět v krátké době testování vůči samotné implementaci. Nutno dodat, že to záleží na povaze projektu, jak moc je produkt sestaven z již odladěných modulů nebo obsahuje nadstandardní funkce a požadavky. Metodika Scrum nedefinuje jak dlouho a jak moc se má testovat, to musí vědět vývojový tým na základě zkušeností. Dohromady uvedené vývojové fáze z tabulky 3.1, činí 85 % z celkového času strávenému na projektu. Zbylých 15 % je rozloženo do: správy konfigurací, která si bere 5 % času a řízení projektu 10 % času. Činnost příprava prostředí se prakticky neprovádí.

c) Analýza procesu vývoje a metodiky Scrum

Pro zmapování procesu vývoje ve firmě Poski a zjištění jaké principy a pravidla firma Poski z metodiky Scrum používá, jsme použili jednoduchý dotazník uzavřených otázek. V prvním sloupci je pravidlo z metodiky Scrum formou otázky a v druhém sloupci, zda je

pravidlo ve firmě uplatňováno. Tento dotazník formou tabulky (tabulka 3.2) byl vyplněn ředitelem výroby.

Otázka	**ano/ne**
Je dodržena transparentnost?	ano
Je definován vlastník produktu?	ne
Je definován ScrumMaster?	ne
Je definován vývojový tým?	ano
Vlastní produktu	
Je zodpovědný za definování položek v produktovém backlogu?	ne
Je zodpovědný za uspořádání těchto položek dle vize produktu a cíle?	ne
Je zodpovědný za kontrolu hodnot a zajištění jejich dodání?	ano
Je zodpovědný za zajištění transparentnosti a dostupnosti produktového backlogu?	ne
Je zodpovědný za zajištění, aby vývojový tým správně tyto položky pochopil?	ano
Je zodpovědný za ROI?	ne
Je respektován vývojovým týmem. Je nezpochybnitelnou autoritou?	ano
ScrumMaster	
Hledá techniky pro efektivnější správu produktového backlogu?	ano
Interpretuje cíle, položky a produktovou vizi vývojovému týmu?	ano
Učí vývojový tým jak vytvářet stručný a zřetelný produktový backlog?	ne
Rozumí a učí jak vytvářet dlouhodobou produktovou vizi v empirickém prostředí?	ano
Rozumí a učí jak aplikovat agilní vývojové principy?	ne
Na schůzkách bývá v pozici moderátora?	ne
Vede vývojový tým směrem k sebe-organizaci a multifunkčnosti?	ano
Učí a vede, jak má vývojový tým vytvářet kvalitní produkty?	ano
Odstraňuje překážky bránicí vývojovému týmu v kvalitní práci?	ne
Koučuje tým k správnému pochopení Scrumu a jeho bezproblémové přijetí?	ne
Plánuje přijetí Scrum v organizaci?	ano
Školí organizaci osvojit a pochopit Scrum a empirický vývoj produktu?	ne
Nalézá změny pro vyšší efektivitu?	ano
Vývojový tým	
Je sebe-organizující?	ne
Je multifunkční?	ne
Zodpovědnost za práci má celý tým?	ne
Vývojový tým neobsahuje další podtýmy?	ano
Činnosti	
Cyklus vývoje softwaru začíná plánovací schůzkou?	ano
Na začátku každého sprintu tým definuje počet vlastností, které zadají do produktového backlog?	ne
Schůzka se skládá ze dvou stejně časově dlouhých částí?	ne

Je vytvořen sprint backlog?	ne
Jsou každodenní schůzky?	ano
Je vyhodnocení sprintu?	ne
Vyhodnocení sprintu trvá 2 – 4 hodiny?	ne
Vlastník produktu prověřuje, jaké práce byly dokončeny?	ano
Vývojový tým projednává problémy ale i úspěchy při vývoji?	ano
Vývojový tým prezentuje výsledky sprintu?	ne
vlastník produktu informuje o aktuálním stavu produktového backlogu a odhaduje datum dokončení?	ne
Scrum tým navrhuje další kroky, které jsou vstupními parametry pro následující plánovací schůzku?	ne
Provádí se retrospektiva sprintu?	ne
Trvá sprint 2 – 4 týdny?	ne
Všechny sprinty mají stejnou dobu?	ne
Během sprintu se neprovádí změna složení vývojového týmu?	ne
Na konci sprintu je provedeno hodnocení, demonstrace softwaru zákazníkovi?	ano
Je produktový backlog vytvářen pomocí uživatelských příběhů?	ne
Hraje se plánovací poker?	ne
Denní schůzka	
Byla položena otázka: Co bylo dokončeno od doby poslední schůzky?	ano
Byla položena otázka: Co bude dokončeno do termínu další schůzky?	ano
Byla položena otázka: Jaké překážky nám stojí v cestě?	ano
Jsou na schůzce lidé rozděleny na prasata a kuřata	ne
Artefakty	
Jsou definovány produktový backlog?	ne
Jsou definovány sprint backlog?	ne
Jsou definovány burndown grafy?	ne

Tab. 3.2 Zmapování pravidel a činností Scrum ve firmě Poski.com

Vlastník produktu není definován, funkci částečně přebírá vedoucí projektu, který však bývá definován jen někdy a to především u velkých projektů. ScrumMaster není definován taktéž, jeho funkci přebírá projektový manažer, ten ale neučí metodiku Scrum, pouze zaštiťuje projekt jako vedoucí. Každodenní schůzky jsou plněny, s každým členem se však schází jednotlivě.

Tak jako u analýzy popisu procesu vývoje softwaru v předcházející kapitole, i zde můžeme na základě položených otázek prohlásit, že firma Poski si z metodiky Scrum převzala jen velice málo pravidel. Konkrétně na padesát čtyři otázek bylo kladně odpovězeno pouze dvacet jedna krát. A na základní otázky typu, zda jsou definovány role a artefakty dle Scrum nebo zda probíhají schůzky a sprinty podle metodiky, bylo odpovězeno záporně.

d) Analýza agilních praktik a principů ve firmě

Protože metodika Scrum se řadí do skupiny agilních metodik. Pokusíme se porovnat agilní praktiky a principy s praktikami a principy ve firmě. Šetření provedeme formou dotazníku a diskuzí nad danými otázkami.

Společnost Poski dává přednost individualitě před nástroji a procesy, iteraci však neupřednostňuje. Firma však klade důraz na fungující software před obsáhlou dokumentací. Bohužel co se týče předností spolupráce se zákazníkem a smlouvami, dává přednost sjednávání smluv. Dále firma dává lehce přednost reakci na změnu a tedy uspokojení zákazníka před plněním plánu. Konkrétní splnění agilních praktik, můžeme vidět v tabulce 3.3 a na další tabulce 3.4 můžeme pak vidět splnění agilních principů ve firmě.

Praktiky	**Splněno**
Přednost individualitě a iteraci před procesy a nástroji.	ano
Přednost fungujícímu softwaru před obsáhlou dokumentací.	ano
Přednost spolupráce se zákazníkem před sjednáváním smluv.	ne
Přednost reakce na změny před plněním plánu.	ano

Tab. 3.3 Splnění agilních praktik ve firmě Poski.com

Principy		**Splněno**
1.	Uspokojení zákazníka díky včasné a průběžné dodávce softwaru.	ano
2.	Vítání požadavků na změny a to i na konci vývoje softwaru.	ne
3.	Průběžně dodávat klientovi funkční software.	ne
4.	Lidé z byznysu a vývojáři pracují spolu denně a to v průběhu celého projektu.	ano
5.	Vedení podporuje potřeby jedinců, motivuje je, důvěřuje jim a zajišťuje jim prostředí pro perfektní práci.	ano
6.	Způsob předávání informací v rámci vývojového týmu a komunikace mimo něj je zajištěn osobní konverzací.	ano
7.	Postup práce se měří fungujícím softwarem.	ne
8.	Všichni, to znamená, sponzoři, vývojáři a uživatelé udržují konstantní tempo rozvoje a podporují trvale udržitelný rozvoj.	ne
9.	Stálá pozornost na kvalitní návrh a technické dokonalosti zvyšující rychlost procesu vývoje.	ne
10.	Jednoduchost, efektivita a zaměření se na opravdu nejdůležitější věci.	ano
11.	Tým je samo-organizující.	ne
12.	V pravidelných intervalech se vyhodnocuje, jak pracovat ještě efektivněji a jak upravit a zlepšit svůj způsob práce.	ne

Tab. 3.4 Splnění agilních principů ve firmě Poski.com

Zákazník je uspokojen pouze včasným dodáním softwaru, průběžné dodávání prototypu se nevykonává. Iterace mezi klientem a týmem je pouze na začátku a na konci vývoje, s klientem neprobíhá kontinuální spolupráce. Před osobním kontaktem mezi klientem a týmem je upřednostňována telefonní nebo emailová komunikace. Tempo vývoje většinou zpomaluje sám klient. Nové požadavky na systém nejsou moc vítány. Vedení se snaží pomáhat jedincům a zkvalitňovat prostředí pro práci. V rámci týmu je osobní komunikace zajištěna. Většinu práce si tým řídí sám, je však kontrolován a veden managementem. Neprobíhá vyhodnocování projektu.

Z porovnání agilních a firemních praktik a principů můžeme vyvodit tyto následující návrhy na změny.

1. Firma Poski by měla častěji spolupracovat a komunikovat se zákazníkem. Po celý průběh délky projektu a nejlépe formou osobní komunikace. Tím by efektivněji reagovala na změny požadavků.
2. Podporovat první návrh na změnu. To znamená snažit se zákazníkovi hned od začátku projektu vysvětlit, že na úspěch projektu má vliv i on. Domluvit si pravidelné schůzky a ujasnit si pravidla komunikace. Definovat úlohy a zodpovědnost klienta.
3. Společnost Poski by měla zavést prototypování, tedy dodávat zákazníkovi fungující meziprodukt. Dodávat tento prototyp průběžně a včas.
4. Přenést zodpovědnost na vývojový tým, aby se sám organizoval. Vedoucí pracovník by měl provádět pouze kontrolu, ne řídit. V metodice Scrum tuto funkci má produktový manažer.
5. V pravidelných intervalech vyhodnocovat, jak efektivněji pracovat. Nejlépe jednou měsíčně. V metodice Scrum vždy po každém sprintu.

3.1.5 SWOT analýza

SWOT analýzu jsme vypracovali společně s managementem firmy Poski. Silné a slabé stránky, příležitosti a hrozby jsme se snažili nalézt pomocí techniky brainstorming, která měla zajistit objektivitu a komplexnost analýzy. Většina sofistikovaných vstupních analýz (například analýza trhu) by byla totiž velmi složitá a navíc bezpředmětné. Cílem SWOT analýzy nebylo nalézt globální problémy firmy a řešit je, ale pouze nastínit pozici firmy na trhu, pochopit politiku firmy, doplnit chybějící aspekty analýzy firmy a pomoci tak při konečném návrhu řešení problémů a implementace metodiky.

Silné stránky:

- velmi dobrý seniorský tým programátorů,
- loajalita managementu a seniorského týmu programátorů,
- zkušenosti, délka působnosti na trhu,
- nadějné produktové portfolio.

Slabé stránky:

- závislost na klíčových pracovnících (seniorské pozice),
- málo expertů, slabá základna junior pracovníků a jejich slabé zkušenost,
- nedostatek času a finančních zdrojů pro potřebné technologické inovace,
- malé portfolio produktu,
- nezkušenost obchodníků vedoucí k špatné specifikaci požadavků od klienta.

Příležitosti:

- využití dobrých referencí,
- spokojenost klientů,
- rozrůstání trhu díky růstu počtu mobilních zařízení,
- využití různých fondů, projektů a dotací jako zdroje kapitálu na inovace, lidské zdroje a vzdělávání zaměstnanců.

Hrozby:

- velikost firmy (firma je v porovnání s jinými společnostmi stále velmi malá),
- snižování cen, především u menších zakázek,
- obrovská konkurence na trhu.

Firma by se měla zaměřit na zefektivnění vývojových procesů. Díky toho zvýší svůj zisk, který může investovat do rozvoje firmy. Po získání kapitálu by firma měla investovat do náboru kvalitních pracovníků, školení svých obchodníků, nových technologií a marketingu. Na základě toho může firma expandovat, růst a být konkurenceschopná menším i větším ostatním firmám na trhu. Pro získání kapitálu a lidských zdrojů může firma využít různých dostupných fondů a spolupracovat s vysokými a středními školami. Zároveň můžeme doporučit zaměřit se vývoj mobilních aplikací a tvorbou webových stránek optimalizovaných na tablety a mobilní telefony (tzv. responsive web design[15]).

3.1.6 Definování problému a závěr analýzy společnosti Poski

Společnost Poski je malá firma (25 zaměstnanců), od toho se odráží i procesy ve firmě, které jsou řízeny neformálně a nejsou stanovena nijak direktivní pravidla. Ekonomická situace společnosti s rostoucím trendem naznačuje, že firma se ubírá správným směrem.

Aktuální stav procesu vývoje softwaru je podobný ostatním stejně velkým firmám. To znamená, nevyvíjí software podle žádné metodiky a procesy ve firmě jsou řízeny nedefinovaně. Podle CMMI je úroveň vyspělosti firmy Repeatable (Opakovatelný), tedy druhý stupeň. Chce-li firma vyvíjet efektivněji a zůstat nadále konkurenceschopná, měla by se pokusit dostat na čtvrtý stupeň vyspělosti – Managed (Řízený) až pátý – Optimized (Optimalizovaný).

Vývojový tým se jevil jako tým expertů a specialistů, kteří ví, jak správně vyvíjejí systém. Na pracovišti během schůzek jsme pociťovali přátelskou atmosféru. Pracovní prostředí nám přišlo příjemné a nenalezli jsme žádné překážky pro efektivní práci zaměstnanců.

Po seznámení se s procesem vývoje softwaru na základě rozhovorů s managementem a zaměstnanci podniku jsme došli k X problémům, ke kterým musíme nalézt návrh na řešení, aby firma Poski mohla efektivně vyvíjet software.

1. Nedokonale a špatně popsány zákazníkovy požadavky. Osoby (nejčastěji obchodníci) zodpovědné za zmapování a zpracování specifikace požadavků vykonávají tyto činnosti nekvalitně. Obchodníci jsou špatně proškoleni a nedostatečně znají a rozumí nabízeným produktům a službám, zároveň nejsou dostatečně seznámeny s technickými aspekty vývoje. Dík špatné specifikace se může zaprvé vytvořit odlišný software, než zákazník žádal, který se musí pak upravovat. Zadruhé cenové ohodnocení práce a návrh časového

[15] Responsive web design, nemá český překlad. Jde o styl vývoje HTML dokumentu, tak aby byl optimalizovaný pro všechny druhy zobrazovacích zařízení.

rámce nemusí být adekvátní s požadovaným produktem, což snižuje potenciální zisk firmy a eventuelní nedodržení termínu.

2. Průběžné neověřování kvality jak ze strany zákazníka, tak ze strany vývojového týmu. Chyba vytvořená na začátku vývoje není ihned nalezena a je „tlačena“ dál. Na tuto špatně naimplementovanou část kódu jsou závislé další části, proto po nalezení a opravení chyby, se musí navazující části opravit také.
3. Dodání softwaru, který nekoresponduje s přáním zákazníka, nedodržení specifikace požadavků nebo nedokonalé zachycení představ klienta. Všechny tyto problémy mají za následek (kromě nespokojenosti klienta) úpravy softwaru, které jsou v konečné fázi vývoje velmi pracné a časově náročné. I v případě perfektně zpracované specifikace systému může nastat odklonění od klientovy vize při implementaci softwaru.
4. Málo častá komunikace se zákazníkem. Kvůli podstatě internetových zakázek, zákazník často mění své požadavky na systém. Tým tak nedokáže flexibilně reagovat na změny v požadavcích, protože tyto požadavky přicházejí až v reakci na předání finálního produktu, tedy v době, kdy systém je již hotový.
5. Nejsou jasně definované zodpovědnosti a kompetence rolí podílející na procesu vývoje SW, včetně zodpovědnosti a kompetence role zákazníka. Díky toho, že osoby neznají své úlohy a kompetence, může nastat opomenutí některé činnosti, nepatřičné delegování nebo nevědomá práce dvou osob na jedné činnosti. Zároveň není popsána zodpovědnost za dílčí konkrétní úkoly na konkrétním projektu. Osoby podílející se na vývoji jsou tak schovány za anonymitou a nepřejímají svoji zodpovědnost.
6. Ve firmě existují přesčasy. Jinými slovy není dodržen osmý princip agilního vývoje – udržujte konstantní tempo rozvoje. Přesčasy a přetěžování pracovníků může krátkodobě vyřešit zpoždění termínu projektu, ale dlouhodobě je zdrojem nízké produktivity práce.
7. Neexistuje samostatný tým technické podpory, mající na starost pouze podporu, opravy a rozvoj již hotových produktů. Některé osoby migrují z role programátora do role technika podpory. To znemožňuje efektivní řízení projektů a znesnadňuje časové plánování, jelikož práce na technické podpoře a servisu jsou nárazové a časově nevyvážené (objemy práce nelze predikovat).
8. Neprovádí se hromadné schůzky. Tým se neschází na hromadných schůzkách za účelem projednání jednotlivých projektů, ale vedoucí výroby se schází s každým členem zvlášť. Tím se ztrácí objektivní pohledy na problémy a synchronizace práce se kompiluje. Zamezuje se tím zlepšení úroveň znalostí každého člena týmu o projektu a společné odstranění problémů.

9. Neprovádí se vyhodnocení a retrospektiva projektu. Tým nemá možnost identifikovat problémy, které nastaly při vývoji a navrhnout kroky na zlepšení. A naopak nemá možnost ani vypíchnout procesy a kroky, které fungovaly dobře, optimalizovat je a zavést je do dalších projektů. Management se tak nedozví kladné a záporné veličiny ovlivňující procesy při vývoji a tím celá firma stagnuje.
10. Chybí jasný, přehledný a transparentní plán vývoje. Chybí seznam funkcí/požadavků, detailní seznam naimplementovaných, otestovaných a neotestovaných těchto funkcí, plus přiřazení konkrétních osob k těmto položkám. Díky toho tým ztrácí přehled o postupu na projektu. Nemají jasný přehled, co mají implementovat a co otestovat. Nejsou si vědomi, v jaké časové fázi se v procesu vývoje nacházejí.
11. Firmě chybí obecně nějaká norma, standard, certifikát či metodika podle, které by mohli efektivně vyvíjet, optimalizovat a řídit.

K zmíněným problémům aktuálního stavu procesu vývoje softwaru doplníme problémy, které znemožnily optimální nasazení metodiky Scrum před dvěma roky, nalezené v kapitole 3.1.3:

12. nevhodnost metodiky Scrum pro jejich aktuální vývoj softwaru,
13. tým nebyl sebe-organizující,
14. tým nebyl multifunkční,
15. špatné odhady časových náročností úkolů,
16. vlastník produktu neměl všechny kompetence, které definuje Scrum,
17. ScrumMaster neměl všechny kompetence, které definuje Scrum.

Na tyto nalezené problémy se pokusíme nalézt řešení v kapitole 4.1. Přičemž některé problémy budou řešeny samotnou metodikou Scrum, některá řešení nalezneme v obecných agilních principech a na zbývající problémy se pokusíme najit odpověď ve firmě D3Soft, kterou budeme analyzovat v další kapitole.

3.2 Analýza společnosti D3Soft

3.2.1 Popis firmy

Společnost D3Soft s.r.o. (dále již jen D3Soft) je středně velká softwarová firma, která se zabývá analýzou, programováním, metodickou a konzultační činností[16]. Působí nejenom v České republice, ale i v dalších státech střední Evropy. D3Soft je součástí skupiny D3Group, ve které jsou kromě společnosti D3Soft Future, zabývající se vývojem informačních systémů, hostování aplikací a implementace ERP systémů, také divize působící v oblasti marketingu a obchodu. Struktura společnosti dokládá diagram organizační struktury v příloze E. Aktuálně má firma D3soft více než 90 zaměstnanců.

Firma D3Soft děli své portfolio do osmi kategorií:

- informační systémy,
- CRM,
- vývoj softwaru na míru,
- Microsoft Dynamics – podniková řešení ERP a CRM,
- Helios – podnikový informační systém, ekonomický a účetní software,
- konzultační služby, zahrnující audit, optimalizaci byznys procesů a projektové řízení rozsáhlých projektů,
- IT služby, například hosting nebo vizualizace serverů,
- e-Commerce, obsahující tvorbu e-shopů a webových stránek, nebo internetový marketing.

Abychom mohli efektivně porovnat proces vývoje ve firmě D3Soft a Poski, budeme se dále zabývat procesem vývoje stejných produktů, tedy vývojem CRM a e-Commerce.

3.2.2 Analýza procesu vývoje softwaru

Společnost D3Soft je na trhu již víc než 16 let, i proto má většinu firemních procesů již optimalizovaných a vyřešených. Všechny projekty jsou řízeny podle firemní projektové metodiky, která zahrnuje organizační zásady, definuje úkoly, povinnosti a kompetence všech rolí (včetně role zákazníka), popisuje samotné projektového řízení, analýzu rizik a řešení konfliktů. Rozebírá a popisuje jednotlivé fáze v procesu vývoje softwaru. Pokrývá

[16] Všechny informace o společnosti byly poskytnuty paní Mgr. Lenkou Mališovou, která působí ve společnosti D3Soft jako produktový specialista.

postimplementační činnosti, jako jsou podpora, servis a rozvoj softwaru. Definuje také různé šablony, jako například testovací scénáře, předávací protokoly nebo zadání pro implementaci.

Tuto projektovou metodiku doplňuje metodika orientovaná konkrétně na vývojové procesy, která pokrývá veškeré aspekty vývoje vycházející z agilních principů a doplněna o nejlepší praktiky na základě zkušeností. Metodika pokrývá celý životní cyklus softwaru včetně předcházející obchodní činnosti. Kromě rozšíření již definovaných rolí, metodika popisuje do detailu vývojové procesy a doporučuje v nich nejlepší praktiky. Metodika dále definuje vstupní a výstupní dokumenty a poskytuje návody k analytickým nástrojům používané při vývoji SW.

D3Soft používá více než 30 definovaných artefaktů, mezi které patří například: zadání pro implementaci, business process model, use case model, testovací scénář, architektura systému, projektový plán, testovací scénář, komunikační matice nebo předání díla.

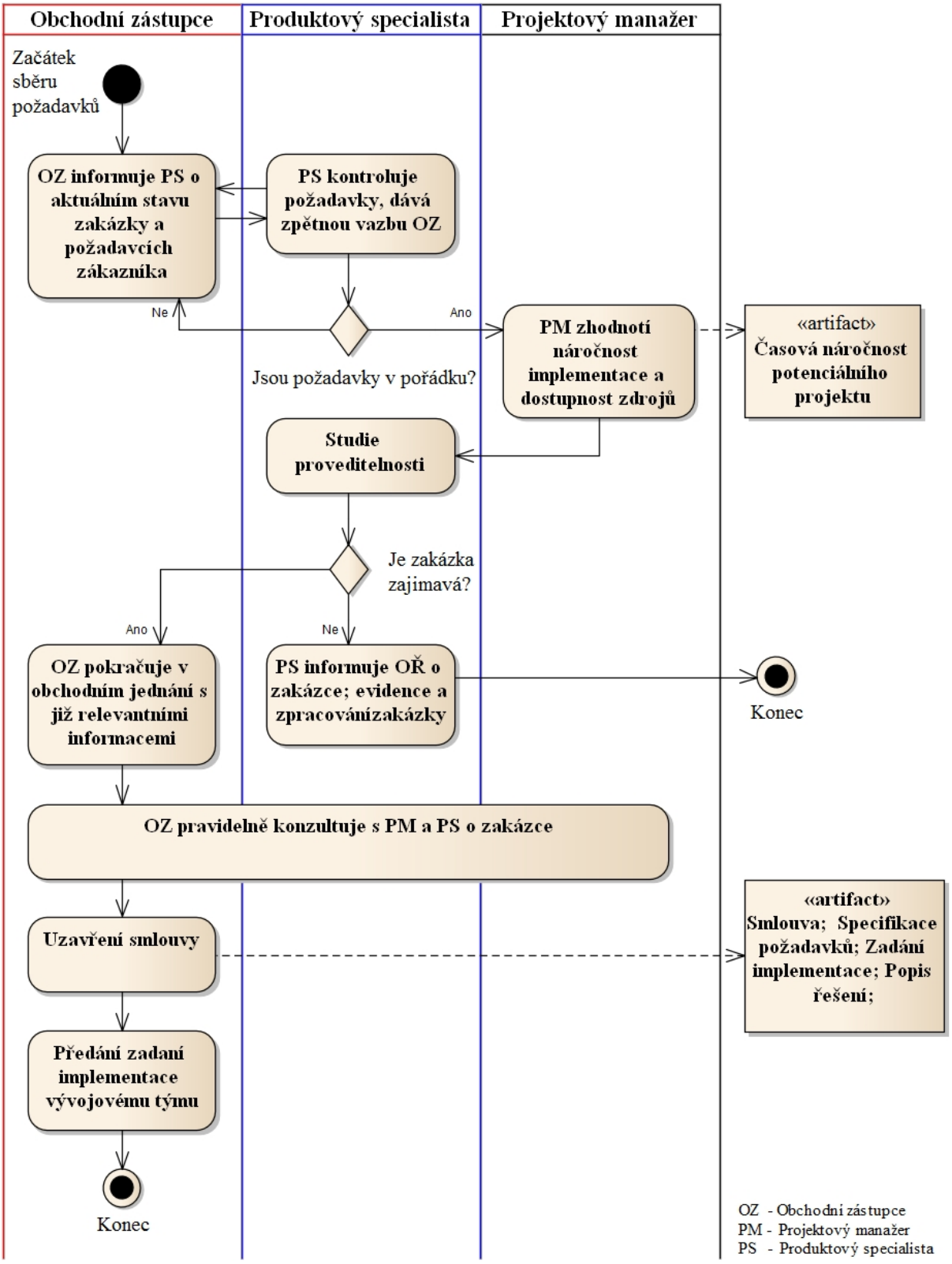
Role ve vývojovém týmu D3soft jsou:

- vedoucí vývoje,
- projektový manažer,
- obchodní zástupce,
- konzultant,
- analytik,
- architekt,
- návrhář uživatelského rozhraní,
- specialista na reporty,
- databázový specialista,
- programátor,
- tester.

Testování probíhá pomocí UNIT testů. D3Soft využívá celopodnikoví informační systém pro podporu organizace a řízení vývoje IT služeb.

Celý proces vývoje softwaru ve firmě D3Soft se proti procesu v Poski značně liší. Největší rozdíl je při zpracování klientových požadavků. Iterací mezi mapováním požadavků a jejím zpracováním je mnohem více než ve firmě Poski. Po zaznamenání požadavků obchodníkem se, i když jsou obchodníci D3Soft velmi dobře technicky vyškoleni, provádí analýza požadavků a odhad náročnosti. Obchodník vytvoří nabídku a po jejím odsouhlasení se vytvoří plán realizace, včetně stanovení konečné ceny a podpisu smlouvy. Poté se přechází

k technické analýze, návrhu a k samotnému vývoji softwaru. Proces před samotným vývojem (zahrnující analýzu, návrh, implementaci a testování) dokládá diagram aktivit (obrázek 3.5). Vývojový proces, pro jeho nemožnost porovnání s procesem v Poski zde uvádět nebudeme.



Obr. 3.3 Diagram aktivit obchodního procesu ve firmě D3Soft

a) Srovnání rozložení jednotlivých fází vývoje softwaru ve firmě Poski a D3Soft

Obdobnou analýzu časového rozložení jednotlivých fází vývoje softwaru jako ve firmě Poski, jsme provedli i ve firmě D3Soft a porovnali v tabulce 3.5.

	Doba trvání v %	
Fáze vývoje softwaru	**Poski.com**	**D3Soft**
Byznys modelování	0	9
Specifikace požadavků	10	10
Analýza	15	25
Návrh	15	15
Implementace	40	25
Testování	10	10
Nasazení	5	5
Vyhodnocení projektu	5	1
Správa konfigurací	5	10
Řízení projektu	10	25
Příprava prostředí	0	10

Tab. 3.5 Porovnání délky fází ve firmách Poski.com a D3Soft

Srovnání obou firem, ať je to z jakéhokoliv hlediska, je velmi obtížné. Firmy nelze srovnat především z důvodu rozdílné velikosti (D3Soft 90 a Poski 25 zaměstnanců) a velikosti realizovaných projektů. Tento rozdíl je vidět především v procentuálním rozložení projektové a vývojové častí. U D3Soft je to rovným dílem 50 % projektová část, kde největší rozdíl lze vidět v porovnání v řízení projektů (25 % D3Sfot, 10 % Poski) a 50 % vývojová část, kde se nejvíc rozchází délky fází implementace (25 % D3Soft, 40 % Poski) a analýzy (25 % D3Soft, 15 % Poski).

b) Srovnání provádění agilních principů ve firmách Poski a D3Soft

Srovnání jsme také provedli na přítomnosti agilních principů ve firmách Poski a D3Soft. Z tabulky 3.6 jde vidět, že firma D3Soft se snaží vyvíjet agilním způsobem. Pouze princip č. 2 „vítání požadavků na změny a to i na konci vývoje softwaru“ se ve firmě těžko uskutečňuje, zvlášť když jde o požadavek, který razantně mění architekturu systému. Jinak záleží na typu produktu, pokud klient chce vyměnit nebo upravit některou funkci, není to pro vývojový tým problém. Pokud, ale klient žádá novou funkci, mění se tím rozpočet i termín dodání produktu.

		Splněno	
Principy		**Poski**	**D3Soft**
1.	Uspokojení zákazníka díky včasné a průběžné dodávce softwaru.	ano	ano
2.	Vítání požadavků na změny a to i na konci vývoje softwaru.	ne	ne
3.	Průběžně dodávat klientovi funkční software.	ne	ano
4.	Lidé z byznysu a vývojáři pracují spolu denně a to v průběhu celého projektu.	ano	ano
5.	Vedení podporuje potřeby jedinců, motivuje je, důvěřuje jim a zajišťuje jim prostředí pro perfektní práci.	ano	ano
6.	Způsob předávání informací v rámci vývojového týmu a komunikace mimo něj je zajištěn osobní konverzací.	ano	ano
7.	Postup práce se měří fungujícím softwarem.	ne	ano
8.	Všichni, to znamená, sponzoři, vývojáři a uživatelé udržují konstantní tempo rozvoje a podporují trvale udržitelný rozvoj.	ne	ano
9.	Stálá pozornost na kvalitní návrh a technické dokonalosti zvyšující rychlost procesu vývoje.	ne	ano
10.	Jednoduchost, efektivita a zaměření se na opravdu nejdůležitější věci.	ano	ano
11.	Tým je samo-organizující.	ne	ano
12.	V pravidelných intervalech se vyhodnocuje, jak pracovat ještě efektivněji a jak upravit a zlepšit svůj způsob práce.	ne	ano

Tab. 3.6 Srovnání agilních principů ve firmách Poski a D3Soft

Tabulka 3.6 dokládá, že D3soft v porovnání s Poski se snaží agilní principy při vývoji uplatňovat více.

3.2.3 Závěr analýzy společnosti D3Soft

Po prostudování firemních metodik a na základě rozhovoru s produktovým specialistou firmy jsem došel k těmto níže uvedeným závěrům. D3Soft vyvíjí iteračním a inkrementačním způsobem. Drží se agilních principů a praktik.

Z procesů, z praktik a ze samotné metodiky jde vidět, že firma se vývojem zabývá již dlouho a většinu procesů mají již optimalizovaných. Podle stupnice CMMI je firma na úrovni páté – optimalizovaný vývoj. Tým podpory je od vývojového týmu personálně odlišný. Tým obchodníků je perfektně technicky vyškolen, rozumí implementačnímu procesu a zná dokonale nabízené produkty. Velmi často spolupracuje s vývojovým týmem a zpřesňuje tak specifikaci požadavků z technického hlediska a plán projektu.

V kapitole 4 nalezené poznatky použijeme v návrhu řešení na nalezené problémy ve firmě Poski a inspirujeme se jimi v konečném návrhu metodiky.

4 Návrh a implementace metodiky podle Scrum

Tato kapitola bude rozdělena do tří částí. V první navrhneme řešení identifikovaných problému z kapitoly 3. V druhé části navrhne celkovou metodiku pro vývoj softwaru ve firmě Poski. V poslední podkapitole bude metodika rozepsána do bodů, jak by se měla implementovat do firmy.

4.1 Návrh řešení identifikovaných problémů

V kapitole 3. jsme zmapovali aktuální stav vývoje softwaru ve firmě Poski.com, nalezli hlavní problémy při vývoji a definovali příčiny nemožnosti nasazení metodiky Scrum ve firmě. Na těchto celkově sedmnáct problémů se pokusíme navrhnout možnosti na řešení. Kromě vlastního pohledu nám bude vzorem především metodika Scrum a agilní principy. Dalším zdrojem nám bude zmapování vývojového procesu ve firmě D3Soft, kde jsme našli inspiraci pro řešení problémů, které Scrum ani agilní principy neřeší. U každého návrhu řešení bude uveden zdroj, kterým jsme se nejvíce v daném problému inspirovali. Závěrem návrhu bude celkový popis procesu vývoje dle metodiky Scrum pro konkrétní potřeby firmy Poski.

Problém:

1. **nedokonale a špatné popsání zákazníkovy požadavky.**

Návrh řešení.

Osoba, která vytváří specifikaci požadavků, musí byt technicky vyškolena a musí rozumět nabízeným produktům. Musí umět co nejpřesněji cenově ohodnotit a navrhnout termín dodání. Obchodník nemá samozřejmě tak hluboké znalosti jako například programátor, takže nedokáže ohodnotit náročnost implementace úplně přesně, proto musí s vývojovým týmem úzce komunikovat a zdokonalovat se. Úlohou obchodníka v rámci vývoje je především dodat jasné a podrobné podklady pro vývojový tým a vlastníka produktu, kteří na základě těchto podkladů vytvoří plán realizace.

Specifikace požadavků a porozumění zákazníkovi je nejdůležitější část projektu, jelikož se v této fázi definuje, co se vlastně bude vyvíjet. Tuto zodpovědnost by měl mít buď obchodník, který jedná s klientem nebo vlastníkem produktu.

Zdroj: Scrum, agilní principy.

Problém:

2. **průběžné neověřování kvality jak ze strany zákazníka, tak ze strany vývojového týmu.**
3. **Dodání softwaru, který nekoresponduje s přáním zákazníka, nedodržení specifikace požadavků nebo nedokonalé zachycení představ klienta.**

Návrh řešení.

Oba tyto problémy řeší iterační a inkrementační vývoj neboli takzvané prototypování. Tento princip vývoje je základem všech agilních metodik a tedy i metodiky Scrumu. Jedná se o to, že firma by měla uvolňovat fungující verze softwaru v několika iteracích. V první iteraci by měla vytvořit prototyp návrhu grafiky, GUI, architektury a struktury. Cílem první iterace by mělo být především ověření, zda vývojový tým správně pochopil klientovy představy a požadavky na systém. To zajistí týmu jasnou vizi o systému a ověření si, že se ubírá správným směrem. Tato první iterace je nesmírně důležitá především v případech, když klient nemá jasnou představu o systému. Koncept softwaru dá tak obou stranám zpětnou vazbu, že se navzájem dobře pochopili.

Další iterace ve vývoji ověřují, zda jednotlivé kroky ve vývoji softwaru korespondují se specifikací požadavků a směřují k požadovanému softwaru. Podmínka, že každý meziprodukt musí být funkční a otestován, zajišťuje minimalizaci závěrečných oprav a výrazně zkracuje konečné testování softwaru. Na konci vývoje je tedy funkční produkt dle požadavků. V metodice Scrum tyto iterace symbolizují jednotlivé sprinty. Ke kvalitě softwaru bude také přispívat prezentace hotových úkolu na denních schůzkách.

Zdroj: Scrum, agilní principy.

Problém:

4. **málo častá komunikace se zákazníkem.**

Návrh řešení.

Vývojový tým musí častěji komunikovat se zákazníkem. A to alespoň na týdenní bázi. Při jakékoliv nejasnosti v požadavcích musí tým kontaktovat zákazníka a probrat daný problém. Cílem každé firmy je především spokojený zákazník, generující zisk. Ve vývoji softwaru tedy není místo pro pochybnosti ve stylu: klient to chtěl asi takhle. Komunikace úzce souvisí s prototypováním, to znamená domluvení termínu dodání meziproduktů a termín zpětné vazby. Komunikaci by měl zajišťovat obchodník nebo vlastník produktu, který v týmu

představuje zájmy klienta. V agilním vývoji se dává přednost osobní komunikaci, pak eventuelně telefonní a nakonec emailové komunikaci.

Zdroj: agilní principy, D3Soft.

Problém:

5. **nejsou jasně definované zodpovědnosti a kompetence rolí podílející na procesu vývoje SW, včetně zodpovědnosti a kompetence role zákazníka.**

Návrh řešení.

Každá osoba si musí být vědoma své úlohy v procesu. Musí znát své kompetence své úlohy a svou pozici na projektu. I když Scrum učí přijímat zodpovědnost za projekt celému týmu. Součásti položky v produktovému backlogu je také údaj, kdo bude položku implementovat, kdo testovat a zda tyto činnosti byli ukončeny. Tím je určena zodpovědnost za každou naprogramovanou položku. Také díky koncepci otázek kladených na denních schůzkách je vždy jasné, kdo na jakém úkolu pracuje a kdo je za něj zodpovědný.

Ve firemní metodice musí být zakomponována definice každé role, podílející se na vývoji. Definice role musí obsahovat obecný popis role, cíl role, její úkoly, zodpovědnost a kompetence. Role by měly být rozděleny na interní (firma) a externí (zákazník).

Obsahem definice role zákazníka může například být zodpovědnost za testování podle testovacích plánů.

Zdroj:Scrum, D3Soft.

Problém:

6. **ve firmě existují přesčasy.**

Návrh řešení.

Agilní metodiky zakazují přesčasy. Odpočinutý zaměstnanec bude pracovat lépe, dělá méně chyb a dokáže se snáze soustředit na svou práci. Ve firmě by měla být striktně dodržena pracovní doba.

Zdroj: agilní principy.

Problém:

7. **neexistuje samostatný tým technické podpory.**

Návrh řešení.

Pokud jde o opravení chyby, nebo malou úpravu systému má se jimi zabývat tým technické podpory. Pouze pokud je požadavek na novou funkci nebo úprava je natolik velká,

že je brána jako nový projekt, měl by se jím zabývat vývojový tým. Je nutné proškolit tým technické podpory a servisu nebo zaměstnat specialisty, kteří se budou zabývat buď vývojem, nebo servisem. Nelze se spoléhat na několik osob, které budou dělat obojí. Zaprvé každý musí být nahraditelný a zadruhé takto nelze efektivně vyvíjet z důvodu časového plánování a efektivní alokaci lidských zdrojů. Zkušení programátoři, kteří dělají obě tyto činnosti, z důvodu jejich hlubokých znalostí musí buď zaškolit začínající spolupracovníky tak, aby je mohli zastoupit, nebo firma musí přijmout nové zaměstnance. Pokud tak z některých důvodů nelze, musejí mít tito senior programátoři pevně stanovenou dobu, ve které se budou servisu věnovat. Tím vlastník produktu dokáže efektivně řídit a alokovat jejich čas pro vývoj. Pokud ovšem chce firma dále růst a vyvíjet správně, měl by být tým technické podpory pevně daný. Jen v případech, pokud objem práce na podpoře bude nižší, mohou pracovníci na podpoře samozřejmě vypomáhat na vývoji. Z dlouhodobého hlediska je však servis a technická podpora konstantní veličinou.

Zdroj: Scrum, D3Soft.

Problém:

8. neprovádí se hromadné schůzky.

Návrh řešení.

Tento problém řeší metodiky Scrum definováním čtyřem typu schůzek. Plánovací schůzka, denní schůzka, vyhodnocení sprintu a retrospektiva sprintu. Všechny tyto schůzky mají společné to, že na ni mají povinnou účast všichni členové celý Scrum tým. Struktura, povinnost docházky, pravidla i náplň schůzky by měla být dodržena podle metodiky Scrum.

Zdroj: Scrum.

Problém:

9. neprovádí se vyhodnocení a retrospektiva projektu.

Návrh řešení.

Za každým vývojovým cyklem by měla proběhnout kromě vyhodnocení také retrospektiva cyklu. Retrospektiva a analýza vývojových procesů by měla provedena také po každém předání produktu zákazníkovi. Jen tak se tým může zlepšovat. Je důležité, aby se těchto schůzek zúčastnili nejenom management, ale i všichni členové vývojového týmu. Management spolu se ScrumMasterem by měli tyto schůzky podporovat a organizovat. Schůzky by měly mít neformální charakter.

Zdroj: Scrum.

Problém:

10. chybí jasný, přehledný a transparentní plán vývoje.

Návrh řešení.

Problém lze vyřešit zavedením dvou artefaktů produktového backlogu respektive sprint backlogu a burndown grafů. Produktový backlog obsahuje seznam požadavků, který je seřazen dle priority, symbolizuje tedy posloupnost kroků, které se musí vykonat. Každá položka obsahuje také údaje, zda je vyřízena, otestována, zodpovědností za obě tyto činnosti a časová náročnost jednotlivých položek. Díky toho existuje jasný plán realizace projektu a přesná identifikace pokroku na projektu. Kromě produktového backlogu lze postup na vývoji využít také burndown grafu, který jej modeluje.

Zdroj: Scrum, D3Soft.

Problém:

11. firmě chybí obecně nějaká norma, standard, certifikát či metodika

Návrh řešení.

Pro tak malou firmu je vývoj podle světově uznávaných standardů či norem zatím de facto zbytečný. Pokud ale chtějí dále růst a být konkurenceschopní, měli by nějaký audit požadovat. Norma, certifikace či standard jim může pomoct především v marketingu, výhody na trhu a akvizici nových zákazníků. V této chvíli bude stačit, když firma začne vyvíjet podle nějaké metodiky, bude svůj vývoj optimalizovat a řídit.

Zdroj: D3Soft.

Řešení na většinu problémů jsme nalezli v metodice Scrum, v agilních principech nebo ve firmě D3Soft. Na problém č. 14 jsme však museli nalézt vlastní řešení a přizpůsobit vývojový proces a metodiku Scrum tak, aby odpovídala vývoji softwaru ve firmy Poski.

Problém:

12. nevhodnost metodiky Scrum pro jejich aktuální vývoj softwaru.

Návrh řešení.

Pro vývoj softwaru podle metodiky Scrum je nutné změnit koncepci procesu vývoje softwaru. Po prostudování odborné literatury a na základě konzultace se zástupcem z firmy D3Soft a emailové korespondence s několika předními metodology a poradci na procesy zaměřující se metodiku Scrum, jsem došel ke čtyřem variantám návrhu úpravy vývoje

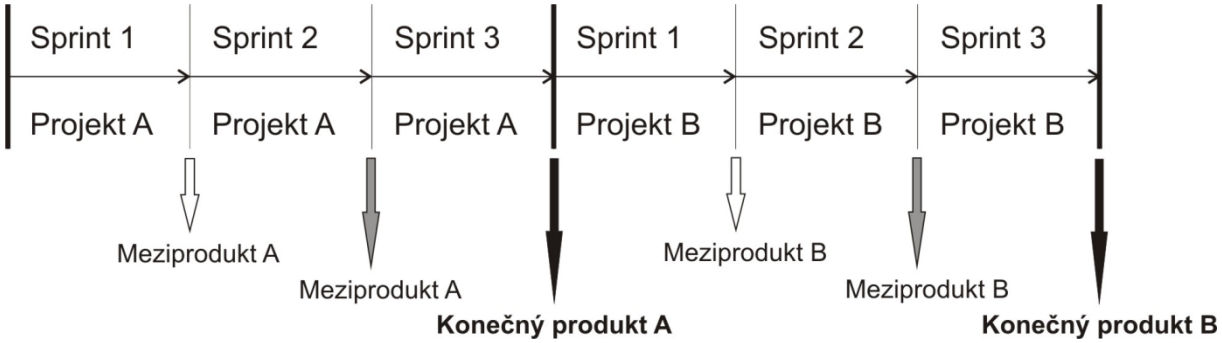
softwaru přizpůsobení metodiky Scrum ve firmě Poski. Základem je rozdělení aktuálně šestnáctičlenného týmu na dva týmy po osmi osobách. Počet sprintu by měl být minimálně tři, u větších projektů může být počet sprintu více.

První možnost.

První možnost je vykonávání sprintů sériově, každý sprint na sebe navazuje a nový projekt se začíná až ve chvíli ukončení projektu prvního a nasazení finálního produktu.

Nevýhodou může být, že tým musí čekat na vyjádření klienta k meziproduktu, což může mít za následek zpomalení vývoje produktu. Novým klientům může vadit, že budou čekat na požadovaný produkt dlouho, na novém projektu se bude pracovat v nejlepším případě, až po ukončení aktuálně běžícího. Na konci projektu vlastník produktu rozhodne, na kterém projektu se bude následně pracovat. Každý projekt má svůj produktový backlog.

První možnost vystihuje obrázek 4.1. Sprinty jsou prováděny po sobě a projekt B se začíná, až po skončení posledního sprintu v projektu A a uvolnění konečného produktu A. Obrázek první možnosti (i možností dalších) zobrazuje velikost všech sprintů konstantních, co samozřejmě nemusí být v reálu pravda. Délka sprintu musí být natolik velká, aby na produktu byl přiměřený přírůstek pro testování a kontrolu.



Obr. 4.1 První možnost vývoje softwaru

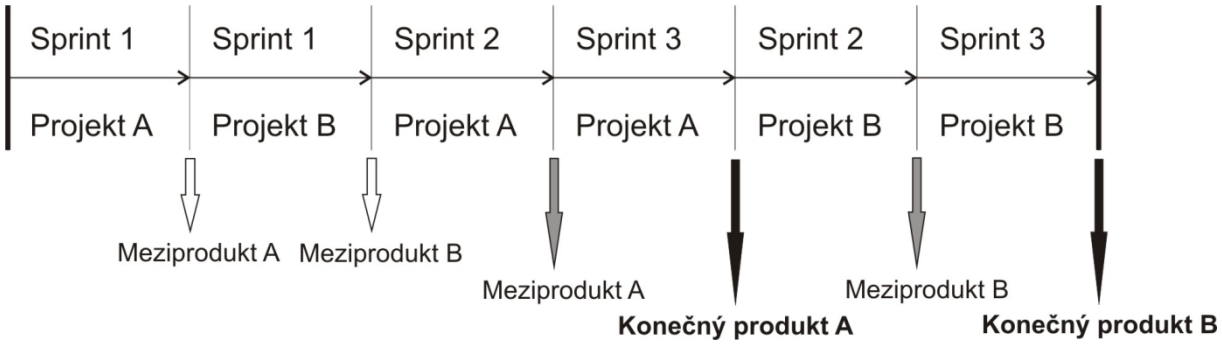
Druhá možnost.

Druhá možnost je vykonávání sprintů taktéž sériově a je velmi podobná první možnosti, s tím rozdílem, že sprinty z různých projektů se mohou prolínat. Na konci každého sprintu vlastník produktu rozhodne, který sprint započne. Nemusí tedy čekat, až se nasadí výsledný produkt. Může například zvolit možnost zahájení nového projektu, z důvodu vyššího potenciálního zisku z projektu nebo důležitosti či velikosti zákazníka. Díky možnosti volby sprintu z jiného projektu nemusí docházet k prodlevám při eventuálním čekání na vyjádření klienta k meziproduktu. Výhoda první i druhé možnosti je, že tým se po určitou dobu (délky sprintu) věnuje pouze jednomu produktu. Nedochází k myšlenkovým přesunům z jednoho

projektu k druhému, vývojáři se mohou snaze soustředit jen na jeden problém a nevytváří se tím chaos. Délka jednotlivých sprintu může být různá, ale vzhledem k typu zakázek firmy a zrychlení vývoje softwaru by měli být sprinty enormně krátké, tedy v rozmezí jednoho až dvou týdnů. U větších projektů může být délka sprintu i tři týdny. I zde má každý projekt svůj vlastní produktový backlog.

Tím, že firma na konci každého sprintu dodá již funkční produkt (i když bez všech požadovaných funkcí), může zvolit platební politiku tak, že bude chtít požadovat zaplacení zlomku konečné ceny na základě objemu prací na produktu a ochránit se tak vůči případným ztrátám.

Druhou možnost popisuje obrázek 4.2, kde v prvním sprintu se vykonávají práce na projektu A, po té vlastník produktu rozhodne o spuštění sprintu v rámci projektu B, následuje znovu projekt A. Na konci druhého sprintu však produktový manažer může například rozhodnou pokračování v projektu B a provede se poslední sprint zakončený uvolnění konečného produktu A. Po té se pracuje již jen na projektu B.

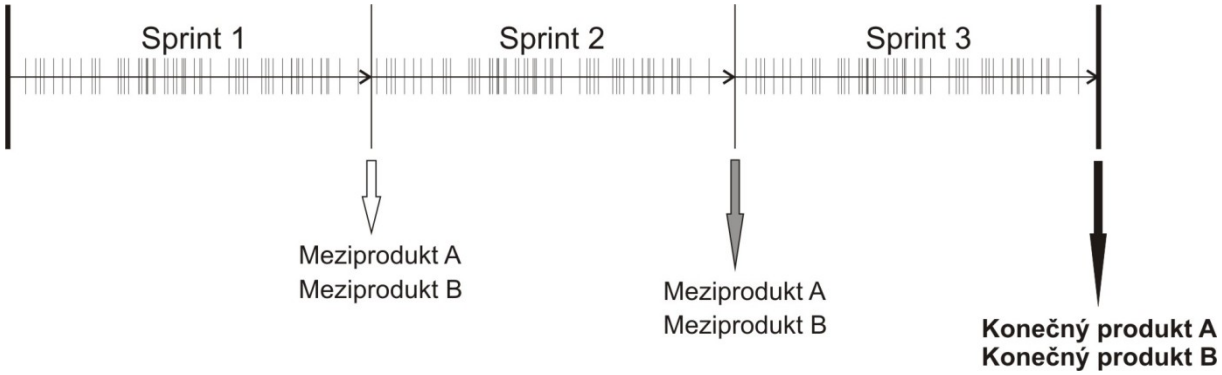


Obr. 4.2 Druhá možnost vývoje softwaru

Třetí možnost.

Třetí možnost je paralelní práce na více projektech, to znamená pracovat v rámci jednoho sprintu na více projektech. Na konci každého sprintu, který je kvůli pracím na více projektech a tedy i většího objemu práce delší, k dispozici tolik meziproduktů, na kolika projektech se v rámci sprintu pracovalo. Největší nevýhodou je, že vývojáři se musí neustále myšlenkově přepínat mezi projekty a může tak docházet k chaosu. Sprinty jsou delší a dodání konečného produktu je tedy také násobně pomalejší. Pro každý projekt je sestaven jeden produktový backlog. Do jednotlivých sprintů (sprint backlogu) je pak vybráno z každého produktového backlogu stejný objem práce/položek tak, aby na konci sprintu byli přírůstky na produktech podobné a mohl být předán meziprodukt s dostatečným počtem funkcí klientovi na otestování a zkontrolování.

Tuto třetí možnost dokládá obrázek 4.3, který ukazuje, že v rámci sprintu 1 se pracuje na obou produktech současně. Na konci sprintu 1 se uvolní oba meziprodukty A i B. Tato činnost se opakuje až do sprintu 3 a uvolnění konečných produktů A, B.

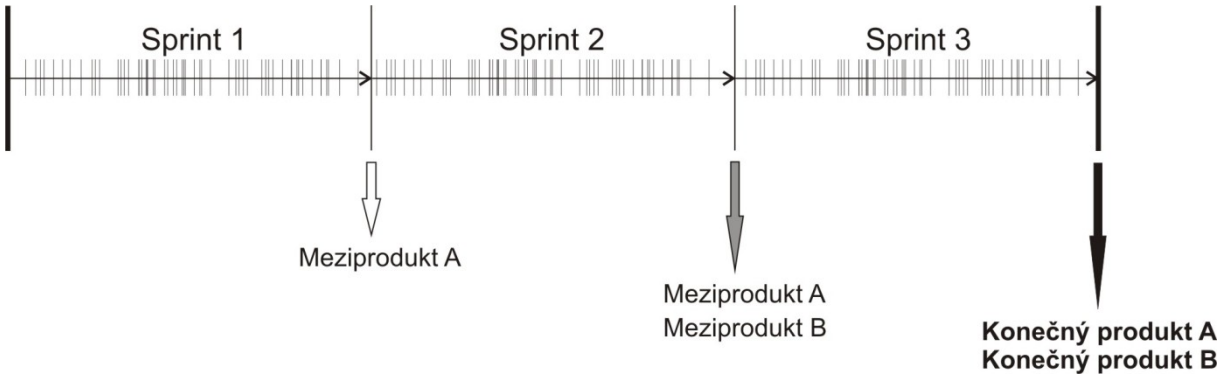


Obr. 4.3 Třetí možnost vývoje softwaru

Čtvrtá možnost.

Poslední, čtvrtá možnost se shoduje s třetí, tedy že v rámci jednoho sprintu se pracuje na více projektech. Rozdíl je v tom, že objem práce na projektech jsou odlišné. Na začátku je totiž sestaven pouze jeden produktový backlog a to pro všechny projekty. Položky v produktovém backlogu jsou sestaveny podle zvolených priorit, kdy každá položka může být z jiného projektu. Položky z produktového backlogu jsou pak vykonávány tak, že ve sprintu se vypracuje prvních několik položek. Na začátku sprintu se podle objemu přírůstku zvolí produkty, které na konci sprintu budou klientovi předány na otestování jako meziprodukty. Tato možnost je z navržených možností nejhorší, protože kromě neustálého přeskakování z projektů na projekt, je tento způsob vývoje nejobtížnější na řízení a správu projektů. Účastníci nemají přehled pokroku na projektech a je velmi obtížné stanovit termín dodání. Bohužel tento způsob výroby je nejblíže podobný aktuálnímu stavu ve firmě.

Obrázek 4.4 zobrazuje čtvrtou možnost, kde v prvním sprintu se pracuje převážně na produktu A a jen minimálně na produktu B. Proto na konci sprintu je uvolněn pouze meziprodukt A, objem práce na produktu B nebyl dostatečný na uvolnění meziproduktu B. Ve sprintu 2 se pracuje na produktu B jen nepatrně více. Objem prací na obou produktech je dostatečný pro uvolnění obou meziproduktu. Ve sprintu 3 se situace opakuje a jsou uvolněny oba konečné produkty.



Obr. 4.4 Čtvrtá možnost vývoje softwaru

Z navržených řešení doporučuji implementovat druhou možnost. Díky jejímu snadnému řízení a jednoduché správě, efektivní práci se zdroji, vývojáři pracují v rámci sprintu pouze na jednom projektu, přesnému návrhu termínu dodání, přesnému zmapování průběhu prací, nemusí se čekat na zákazníka a vývoj SW probíhá konstantním tempem.

Problém:

13. tým nebyl sebe-organizující.

Návrh řešení.

Vývojový tým bude pracovat efektivně, pokud bude sebe-organizující. Aby byl sebe-organizující, musí zodpovědnost projektu být na celém týmu, jeho rozhodnutí tak bude mít váhu. Jediný vedoucí v týmu, může být nazván vlastník produktu, ten totiž apeluje na tým, které požadavky mají být prioritně vyřízeny – co bude provedeno. Sestavení a seřazení těchto požadavků v produktovém backlogu však provedl tým. Vlastník produktu tedy pouze ukazuje směr, který si tým na plánovací schůzce sám zvolil. Tým se bude sebe-organizovat jak během plánovací schůzky, tak během samotného sprintu (sebe-organizuje se na denní bázi na denních schůzkách), díky toho přebere zodpovědnost za provedení práce – bude se rozhodovat, jak provede určitou práci. K sebe-organizaci učí tým ScrumMaster. Management firmy kontroluje tým na schůzce vyhodnocení sprintu, kde přihlíží, jak probíhají práce na projektu.

Zdroj: Scrum.

Problém:

14. tým nebyl multifunkční.

Návrh řešení.

Tým nebyl multifunkční, protože v týmu chyběla role grafika. Firma proto musí zaměstnat grafika na plný úvazek, aby byl dostupný kdykoliv. Pokud se firma rozhodne, že by grafik nebyl natolik vytížen, musí s externím grafikem nastavit taková pravidla spolupráce, aby tým nemusel na jeho práci čekat.

Problém, že někteří členové vývojového týmu suplují také na technické podpoře a servisu, jsme již vyřešili v problému číslo 7.

Zdroj: Scrum, agilní principy, D3Soft.

Problém:

15. špatné odhady časových náročností úkolů.

Návrh řešení.

Scrum se vyznačuje jako jedna z nejlepších agilních metodik pro časový odhad. Největší zásluhu na to mají dvoje činnosti. Za prvé hromadné plánovací schůzky, které přispívají k objektivnějšímu a přesnějšímu odhadu náročnosti úkolů, díky více pohledů na danou problematiku. Druhou činností pro správné a přesné odhady časových náročností úkolu je plánování pokerem, což je technika, která využívá hravou formou právě více pohledů na věc. Zároveň nutí tým o problému diskutovat. Výstupem odhadů a mapování průběhu projektu jsou burndown grafy, které jsou zároveň vstupními daty pro odhady termínu dalších projektů.

Přesnějších prvotních odhadům termínů dodání a poskytnutí je zákazníkovi na začátku projektů, docílí firma také tím, že bude mít vysoce proškolený profesionální tým obchodníků, kteří budou úzce spolupracovat s vývojovým týmem respektive s vlastníkem produktu. Obchodníkům jsou také k dispozici burndown grafy. Firmě se bude tak lépe komunikovat s klientem v průběhu projektu, protože už od začátku se bude pracovat reálnými termíny.

Zdroj: Scrum. D3Soft.

Problém:

16. vlastník produktu neměl všechny kompetence, které definuje Scrum.

Návrh řešení.

Je nutné definovat roli vlastník produktu, který bude mít zodpovědnost za korektní vyřizování klientových požadavků.

Vlastník produktu by měl být zvolen obchodník, konzultant, produktový manažer nebo někdo z marketingového oddělení. Musí mít obchodní pohled na projekt a výborné komunikační schopnosti, jelikož je to právě on, kdo nejčastěji komunikuje se zákazníkem a zajišťuje co nejvyšší výnos ze zakázky. Podmínkou jsou však také jeho hlubší technické znalosti, aby porozuměl také vývojovému týmu. Detailnější popis kompetencí a funkce vlastníka produktu jsme definovali v teoretické části.

Zdroj: Scrum, D3Soft.

Problém:

17. ScrumMaster neměl všechny kompetence, které definuje Scrum.

Návrh řešení.

Stejně jako definování osob na pozici vlastníka produktu i pozice ScrumMastera je nutná, musí být obsazena. ScrumMaster musí mít přirozenou autoritu, protože tým trénuje a školí. ScrumMasterem by měl být zvolen projektový manažer, který není nadřízeným vůči týmu, ale má autoritu týmu.

Bohužel málokterá firma si může dovolit speciální funkci metodika pro optimalizaci procesu, kterým ScrumMaster je. Proto firma může využít spolupráce s vysokými školami, nabídnout tuto funkci jako placenou i neplacenou stáž, eventuelně formou brigády. Dát tak možnost studentům pro získání praxe a zároveň značně ušetřit náklady. Student v této pozici sice nebude mít autoritu a zkušenosti se školením zkušenějších spolupracovníků, kterou metodika Scrum a obecně poradenství doporučuje. Pro kontrolu a dodržování pravidel podle metodiky Scrum to bude stačit. Při zavedení metodiky je však potřeba roli ScrumMastera zadat někomu z managementu nebo zkušenému externímu metodikovi, kteří budou mít potřebnou autoritu pro zavedení nových postupů a procesů do firmy. Podrobnou charakteristika a úlohy ScrumMastera jsou popsány v teoretické části.

Obsahem další kapitoly bude začlenění řešení těchto návrhů do metodiky.

4.2 Návrh metodiky

Při popisu návrhu metodiky se budeme držet metodiky Scrum, tedy pouze procesního rámce. To znamená, že nebudeme dopodrobna popisovat všechny kroky a činnosti při vývojovém procesu. Limitováni jsme také rozsahem, odpovídající diplomové práce. Pouze v případě některých klíčových činností budeme zabíhat do detailu, které budou inspirovány praktikami ve firmě D3Soft. V návaznosti na velikost firmy a složitost softwaru je navíc nežádoucí nasazovat příliš objemnou a složitou metodiku. Navržená metodika tedy kromě stručně popsaného procesního rámce Scrum (podrobnější popis obsahuje teoretická část) bude obsahovat soubor doporučení, nejlepších praktik a agilních principů. Součástí navržené metodiky jsou také návrhy na řešení problému z minulé kapitoly a navržení nového diagramu aktivit (obrázek 4.5).

Role při vývoji softwaru:

- zákazník,
- ScrumMaster,
- vlastník produktu,
- vývojový tým, složen z analytiků, konzultantů, architektů, grafiků, programátorů, databázistů a testerů,
- stakeholders (zúčastněné strany) – vedení firmy, akcionáři, lidé z jiných projektů.

Na managementu firmy je povinností určit jasně přiřazené zodpovědnosti určitým rolím a konkrétním osobám. Především role zákazníka a konkrétní role ve vývojovém týmu. Popis rolí bude mít strukturu: cíl role, úkoly, zodpovědnost a kompetence role. Popis rolí ScrumMastera, vlastník produktu a obecně vývojového týmu je součásti metodiky Scrum.

Vývojové artefakty[17]:

- produktový backlog,
- Sprint backlog,
- burndown graf uvolnění a sprint brundown graf,
- testovací scénář.

Aktuální šestnáctičlenný vývojový tým bude rozdělen do dvou týmů po osmi. Bude definován tým technické podpory a správy, který nebude spadat pod vývoj. Struktura návrhu nové metodiky vývoje softwaru bude rozdělena do několika fází, které jsme si definovali

[17] Projektové artefakty jakou jsou smlouva, nabídka nebo předávací protokoly si firma musí definovat sama.

v teoretické části a pracujeme s nimi v celé této práci. Jak již jsme uvedli v úvodu, celý navržený proces bude zaznamenán do diagramu aktivit na konci této kapitoly (obrázek 4.5). V obrázku není z důvodu přehlednosti zakresleno, že burndown graf a sprint backlog jsou vstupními a výstupními artefakty pro veškeré činnosti ve sprintu. To samé platí pro produktový backlog, ale v rámci celého procesu.

a) Byznys modelování.

Momentálně firma tuto fázi nemusí provádět. Vzhledem k velikosti a typů projektů, je byznys modelování nepotřebné a pouze by vývoj softwaru brzdilo. S růstem firmy a získáním větších zakázek bude byznys modelování ve vývojovém procesu již zapotřebí.

b) Specifikace požadavků.

Klíčová fáze. Obchodník a vlastník produktu musejí projít školeními na seznámení s implementací, znalosti nabízených produktů a získání dovednosti správného sestavení detailní specifikace požadavků. Zpracování požadavků je zobrazeno pomocí diagramu aktivit na obrázku 4.5.

Bude vytvořena jednotná velmi podrobná šablona na specifikaci požadavků, kterou bude součásti soubor otázek, tak aby bylo získáno co možná nejvíc informací o požadovaném produktu. Interní části šablony bude obsahovat návrhy cen pro jednotlivé druhy funkcí a požadavků. Obchodníci budou odměňováni formou části procentem z vygenerovaného zisku (ROI), ne z procenta prodaného produktu. Šablona bude vytvořena vývojovým tým, včetně managementu.

c) Analýza.

Vstupním dokumentem analýzy bude specifikace požadavků. Výstupním artefaktem bude produktový backlog, v dalších etapách procesu se bude používat už jen on.

Součástí analýzy je plánovací schůzka dle Scrumu. Zároveň zde začíná Scrum proces. Součástí plánovací schůzky je tak tvorba projektového plánu obsahující projektový rozpočet, odhady nákladů a potřeb, návrh ROI, délku projektu a plán uvolnění produktu. Je sestavena vize projektu a vytvořen produktový backlog. Z něj je vytvořen sprint backlog, definován cíl sprintu a začíná první sprint – návrh. Obsahem sprint backlogu je samozřejmě časové ohodnocení, které je provedeno technikou plánovaní pokerem.

d) Návrh.

V této fázi proběhne první sprint, v rámci kterého se vytvoří graficky návrh, architektury systému a datový model. Ze všech pozdějších sprintů bude tento sprint nejkratší. U menších projektů může trvat třeba jen dva nebo tři dny. Je ale nutné aby tento sprint proběhl se všemi náležitostmi definovaných Scrumem. To znamená denní schůzky (plánovací schůzka a vytvoření sprint backlogu proběhlo ve fázi analýzy), samotný sprint, vyhodnocení sprintu – prezentace konceptu zákazníkovi, aktualizace burndown grafů a krátká retrospektiva sprintu. Výsledný meziprodukt, tedy koncept bude sloužit ke kontrole klientových požadavku, pro schválení grafiky a návrhu uživatelského prostředí, kontrole vize a směru projektu. Po odsouhlasení konceptu začíná nový sprint, v opačném případě se vytvoří novy návrh produktu.

e) Implementace.

Ve fázi implementace proběhne samotná realizace projektu, ve které proběhnou minimálně dva sprinty. Sprint bude obsahovat plánovací schůzku generující cíl sprintu a sprint backlog. Následovat bude samotný sprint, který bude obsahovat implementaci přírůstku na produktu, testování přírůstku a pokud to bude u klienta možné nasazení tohoto meziproduktu u klienta nebo do testovacího provozu. Na závěr proběhne vyhodnocení a retrospektiva sprintu podle jak definuje metodika Scrum.

f) Testování.

Jelikož se testovalo na konci každého sprintu ve fázi implementace, opětovné testování není nutné. V případech velkých komplexních systému a u důležitých projektů může proběhnout generální testování. Firma by měla začít testovat pomocí UNIT testů.

g) Nasazení.

Nasazení nebo předání produktu bude obsahovat také otestování kompatibility s ostatními systémy, pokud tak nebylo možné učinit v jednotlivých sprintech. Součástí předání bude schválení produktu, školení a nakonec ukončení vývojové části projektu.

h) Udržování systému, rozvoj systému a stažení systému.

O problematiku post implementační části se stará tým technické podpory. Na tuto problematiku zaměřují speciální metodiky a my ji v této fázi nebudeme řešit. Nastíníme zde řešení pro vyřizování požadavků.

Pokud přijde nový požadavek na produkt během vývoje, je připsán do produktového backlogu a upravena cena vyhotovení. Pokud přijde od zákazníka požadavek na změnu, je

vítána a software se náležitě upraví stejně tak produktový backlog a cena za vyhotovení. Pokud klient nalezne nesrovnalost se specifikací, která je součástí smlouvy, software je upraven bez navýšení ceny. Je důležité si s klientem přesně a striktně nastavit pravidla těchto žádostí. Specifikace požadavků musí být součástí smlouvy o vytvoření softwarového díla.

Pokud přijde požadavek na úpravu softwaru po nasazení a předání softwaru, tým technické podpory rozhodne, zda jde u již předaného konečného produktu o malou chybu nebo malou úpravu softwaru, pakliže ano, požadavek vyřizuje tým technické podpory. Nebo jde o velkou chybu aplikace, kterou podpora nedokáže sama opravit, nebo jde o velkou úpravu softwaru (třeba přidání nové funkce), pak požadavek vyřizuje vývojový tým. Pokud nastane spor o vyřízení požadavku, nebo tým podpory nedokáže požadavek vyhodnotit, o jejím zpracování rozhodne vlastník produktu nebo vedoucí týmu podpory. I v případě post implementačních úprav je nutné nastavit ceny těchto úprav, stejně tak rychlost reakce odstranění chyb.

i) Vyhodnocení projektu.

Vyhodnocuje se po každém sprintu. Na závěr celého projektu proběhne ještě celkové vyhodnocení a ohlédnutí se za projektem. Vyhodnotí se průběhu projektu, plnění cílů, termínů a zisk z projektu, ze kterého se vypočítají odměny pro obchodníky a vlastníka produktu. Definují se problémy a diskutuje se, co udělat příště lépe, aby tvorba softwaru byla efektivnější a generovala vyšší zisk. Této schůzky mají povinnost se zúčastnit všichni členové Scrum týmu a management podniku.

Následují tři činnosti napříč celým procesem vývoje softwaru.

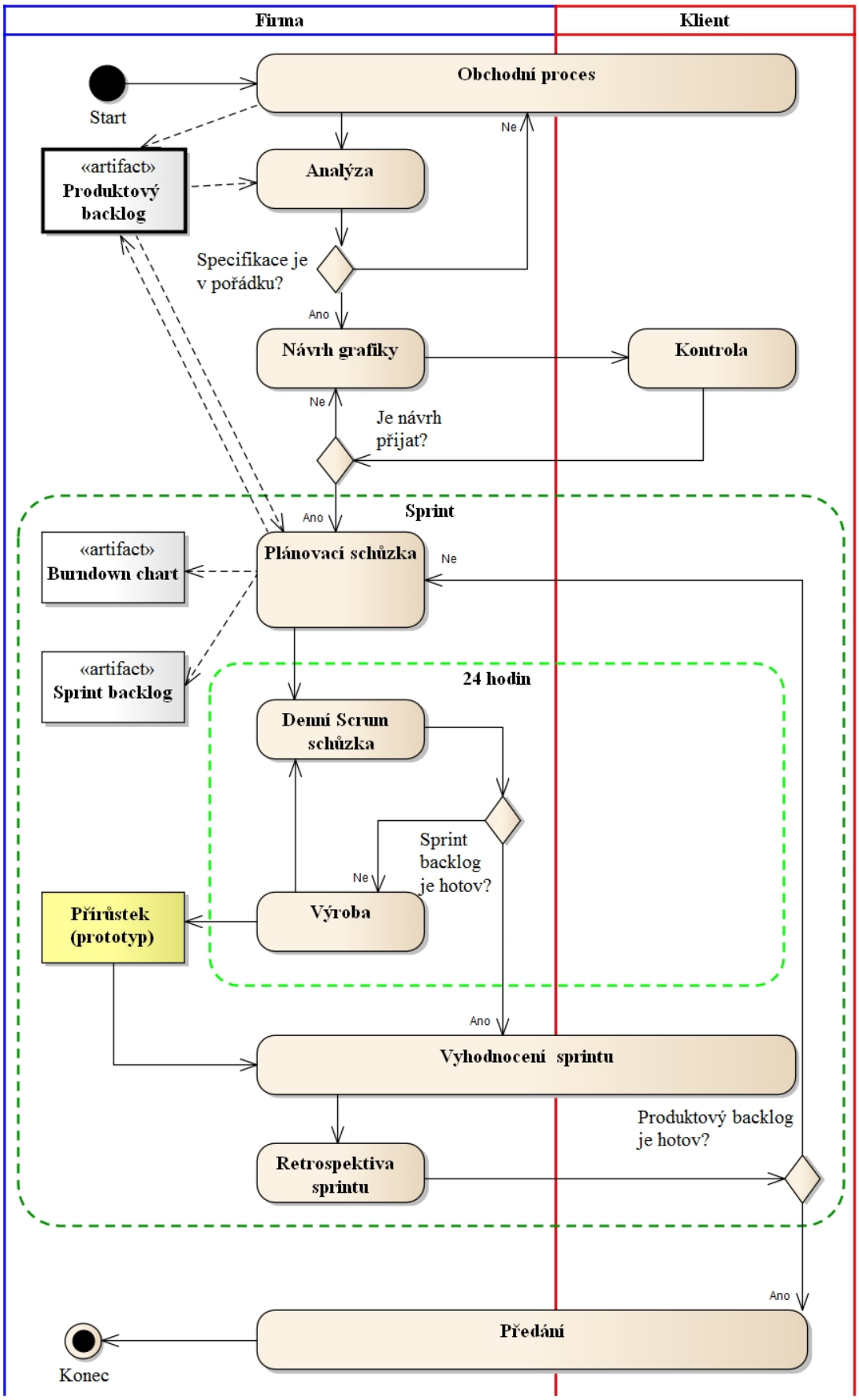
j) Správa konfigurací řízení projektu a příprava prostředí.

Ve správě konfigurací je nutné zajištění především nezbytných artefaktů pro vývojáře i tým podpory. Politiku rozvoje a udržování systému jsme již navrhli.

Projekt řídí samotný vývojový tým. Pouze vlastník produktu kontroluje průběh prací a přijímá zodpovědnost za ROI.

Přípravu prostředí má na starosti konzultant, analytik, architekt a zákazník. Tyto osoby musejí připravit patřičným způsobem prostředí tak, aby produkt byl bez problému nasazen a to jak hardwarové tak organizační prostředí.

Na všechny tyto projektové činnosti by měl být kladen čím dál větší důraz s návazností na růst firmy.



Obr. 4.5 Diagram aktivit navrženého procesu vývoje SW

4.3 Implementace metodiky

Kapitola bude obsahovat seznam kroků pro implementaci navržené metodiky, jak by se měly chronologicky aplikovat do firmy.

1. Definovat a proškolit ScrumMastera nebo přijmout specialistu na tuto pozici.
2. Vytvořit plán nasazení, včetně určení data spuštění vývoje dle metodiky.
3. Seznámit celou firmu o projektu nasazení nové metodiky.
4. Vytvoření týmu a definování rolí.
   4.1. Definování všech rolí (popis, zodpovědnost, kompetence, úkoly).
   4.2. Vytvoření samostatného týmu podpory.
   4.3. Zajistit multifunkčnost týmu.
      4.3.1. Zajistit dostupnost grafika.
      4.3.2. Zajistit technické a produktové školení obchodníků.
   4.4. Určit dva vlastníky produktu.
   4.5. Rozdělit tým do týmu po osmi členech.
5. Vytvoření artefaktů.
   5.1. Vytvoření transparentní dokumentace metodiky.
   5.2. Vytvořit jednotnou komplexní šablonu pro zpracování specifikace požadavků.
   5.3. Vytvoření šablony pro produktový backlog.
   5.4. Vytvoření šablony pro burndown grafy.
   5.5. Zakoupit nebo vyrobit karty pro techniku plánování pokerem.
6. Provést školení všech pracovníků na navrhovanou metodiku.
   6.1. Proškolit vlastníka produktu.
   6.2. Proškolit tým na navrhovanou metodiku.
   6.3. Prezentovat nové řešení vývoje – vývoj v iteracích
   6.4. Prezentovat nové řešení servisních požadavků a technické podpory.
   6.5. Naučit tým techniku plánování pokerem.
7. Výběr vhodného počátečního projektu.
8. Aplikovat metodiku na projekt.
9. Metodický pohled, zajištění podpory a pravidelná školení na metodiku v rámci prvního projektu.
10. Vyhodnocení počátečního projektu z hlediska metodiky.
11. Plošné zavedení metodiky do celé firemní struktury.

Při prvním nasazení a posléze i plošném aplikování metodiky ve firmě je nezbytně nutné přijmout metodiku komplexně. Všechny navržené procesy a pravidla na sebe úzce navazují a jsou na sobě závislé. Vynechání některé činnosti bude mít za následek neúspěšné přijetí metodiky.

5 Závěr

V diplomové práci jsme se zabývali problematikou procesu vývoje softwaru a řízení softwarových projektů. Na problematiku jsme pohlíželi především z pohledu metodiky Scrum a obecně z pohledu agilních metodik, což je skupina metodik do které Scrum patří.

Primárním cílem práce bylo navrhnout a implementovat metodiku dle Scrum do softwarové firmy. Touto firmou byla malá softwarová firma Poski.com, zabývající se tvorbou internetových systémů, aplikací a webových prezentací.

Postup k tomuto cíli byl chronologicky shodný s uspořádáním kapitol v této práci. Základem bylo nastudovat teorii agilních principů a praktik a samozřejmě do detailu pochopit celou strukturu metodiky Scrum a znát všechna její pravidla. Jak naznačuje výčet liberálních pramenů, zdrojů k této metodice existuje mnoho. Tuto skutečnost jsme tedy plně využili a metodiku Scrum prostudovali opravdu do detailů.

Po nastudování teorie jsme přešli k analýze společnosti Poski.com. Z tohoto důvodu jsem do firmy docházel na konzultace a komunikoval s jejich výrobním ředitelem. Abychom dokázali navrhnout optimální metodiku pro jejich potřeby, bylo nutné poznat firmu ze všech aspektů a dopodrobna zmapovat jejich výrobní proces. Provedli jsme tedy analýzu všech postupů a procesů. Nalezení problému jsme provedli formou porovnání agilních a Scrum principů s principy ve firmě. V analýze jsme nalezli a definovali 17 problémů, které jsou ve vývojovém procesu obsaženy a znemožňují tak efektivně vyvíjet kvalitní software. Firma nevyvíjela software podle žádné metodiky nebo definovaných postupů. V minulosti se pokoušela nasadit právě metodiku Scrum, ale bezúspěšně.

Než jsme mohli přejít k návrhu metodiky, bylo nutné nalézt odpovědi na problémy, které metodika Scrum ani agilní principy neřeší. Proto jsme provedli analýzu další softwarové firmy D3Soft. D3Soft je střední softwarová firma, která již vyvíjí software agilním způsobem. Vývojový proces mají definovaný firemní metodikou, kterou na základě zkušeností a nejlepších praktik dále optimalizují. Do této firmy jsem taktéž docházel a na základě rozhovorů s produktovým specialistou firmy jsem hledal řešení na problémy nalezené ve firmě Poski.com. Analýza vývojového procesu byla nastavena především z pohledu porovnání s firmou Poski.com a nalezení tak optimálních principů a zásad, vhodné pro aplikování do navrhované metodiky.

Abychom mohli tak učinit, bylo nutné pochopit komplexně celý vývojový proces softwaru včetně organizační struktury a procesů uvnitř firmy. Analýza firmy v této práci pak obsahuje pouze zlomek potřebný pro doložení nalezených řešení. Není tedy tolik obsáhlá jako

ve firmě Poski.com. Po získání všech potřebných informací jsme provedli návrh řešení pro nalezené problémy. U každého problému jsme uvedli zdroj inspirace, zda byl problém vyřešen díky metodiky Scrum, obecných agilních principů nebo na základě principů z firmy D3Soft. Zásadním problémem bylo přizpůsobit vývojové iterace ve firmě Poski.com dle metodiky Scrum. Zde nám kromě výše uvedených zdrojů byly také emailové korespondence s několika předními metodology a poradci na projektové řízení zabývající se agilními metodikami nebo speciálně metodikou Scrum. Také na základě těchto informací jsme došli ke čtyřem návrhům, jak uzpůsobit iterace ve vývojovém procesu. S návazností na reálné využití těchto návrhů ve firmě Poski.com a maximální efektivitu jsme doporučili návrh sériového vykonávání iterací. Poté jsme provedli shrnutí všech navržených řešení a rozdělili dle použití do jednotlivých fází životního cyklu softwaru. Na závěr jsme vytvořili diagram aktivit celého navrženého vývojového procesu. Základem navržené metodiky je iterační a inkrementační vývoj softwaru. Na závěr jsme vytvořili návod, jak navrhovanou metodiku implementovat do firmy. Primární cíl této diplomové byl, jak lze vidět také z popisu postupu, splněn.

Můj druhý vytyčený cíl bylo reálné využití konečného návrhu v praxi. Jak jsme uvedli v úvodu práce, tento cíl nelze posoudit v termínu odevzdání práce, ale až po nějaké době na základě vyhodnocení aplikovaných změn. Věřím ale, že díky správné analýzy a zachycení všech potřeb firmy Poski.com, jsem dospěl k návrhům, které jsou použitelné v praxi. Kvalitu návrhů zajišťují aplikování agilních principů, převzetí již osvědčených principů z firmy D3Soft a samozřejmě samotná metodika Scrum. Reálné využití všech navrhovaných změn jsme navíc konzultovali s výrobním ředitelem firmy Poski.com.

Diplomová práce, je rozsáhlejšího charakteru z důvodu zvolení tématu z oblasti metodik, které jsou samo osobě rozsáhlé a samotné popsání metodiky Scrum zabralo velkou část práce. Dalším důvodem obsáhlosti práce je mé rozhodnutí, ponechat obrázky, tabulky a diagramy v textu, pro zachování plynulosti a srozumitelnosti obsahu.

Literatury a vědecké práce zpracovaných na toto téma v českém jazyce není hodně. Navíc díky nejednotné terminologii a možná i špatného překladu nejsou dle mého názoru kvalitně sepsány. Rád bych proto věřil, že mé podrobné zpracování metodiky Scrum a přizpůsobení metodiky pro malou firmu bude námětem a zdrojem inspirací pro české, ať komerční, či nekomerční vývojové týmy.

Dle mého názoru by se totiž mělo na problematiku procesu vývoje softwaru z hlediska efektivity daleko více soustředit než je tomu dnes. Především české firmy by si měly konečně uvědomit, že uvolnění finančních zdrojů pro optimalizaci výroby, či dodávky služby vedou

k navýšení zisku mnohem rychleji, než obrácený princip. Většina firem totiž optimalizuje výrobní proces až po vygenerování nějakého většího zisku a jakéhosi postavení firmy na trhu.

I proto se chci touto problematikou dále zabývat a mé další kroky budou vést právě směrem k zefektivňování procesů vývoje softwarů pomocí metodik a projektového řízení obecně.

Použité nástroje

Použité nástroje při tvorbě této práce byly: Microsoft Word, Microsoft Excel, IrfanView, Enterprise Architect (Sparx Systems). Cizí text byl přeložen pomocí Překladače Google dostupný na URL: http://translate.google.cz.

Seznam použité literatury

Knihy a ostatní tištěné zdroje

ARLOW, Jim a Ila, NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: Objektově orientovaná analýza a návrh prakticky.* Brno: Computer Press, 2007. 567 s. ISBN 978-80-251-1503-9.

BAAROVÁ, Jana. *Tvorba platformy podnikové spolupráce pomocí metodiky SCRUM.* Brno, 2012. Diplomová práce. Masarykova univerzita, Fakulta informatiky.

BECK, Kent. *Extrémní programování.* Vydání první. Praha: Grada Publishing, 2002. 160 s. ISBN 80-247-0300-9.

BOEHM, Barry W. *A Spiral Model of Software Development and Enhancement.* TRW Defense Syst. Group, Redondo Beach, CA, roč. 21, č. 5, August 1988. ISSN: 0018-9162.

BRITTON, Carol a Jill DOAKE. *A student guide to object-oriented development.* Amsterdam: Elsevier Butterworth-Heinemann, 2005. 416 s. ISBN 978-075-0661-232.

BROWN, Alan W., David J. CARNEY, Edwin J. MORRIS, Dennis B. SMITH a Paul F. ZARRELLA. *Principles of CASE Tool Integration.* Oxford: Oxford University Press, 1994. 288 s. ISBN 978-01-950-9478-7.

BUCHALCEVOVÁ, Alena a M. LEITL. *Průzkum používání agilních metodik v ČR.* Praha 23.11.2006 – 24.11.2006. In: Objekty 2006. Praha : PEF ČZU, 2006, s. 125–136. ISBN 80-213-1568-7.

BUCHALCEVOVÁ, Alena. *Metodiky vývoje a údržby informačních systémů.* Praha: Grada publishing, 2005. 164 s. ISBN 80-247-1075-7.

CARTLIDGE, Alison, Ashley HANNA, Colin RUDD, Ivor MACFARLANE, John WINDEBANK a Stuart RANCE. *Úvodní přehled ITIL V3.* itSMF, 2007. 58 s. ISBN 0-9551245-8-1.

COCKBURN, Alistair. *Crystal Clear: A Human Powered Methodology for Small Teams.* Boston: Addison-Wesley. 2004. 336. ISBN 978-0201699470.

COCKBURN, Alistair. *Use Cases : Jak efektivně modelovat aplikace.* Praha: Computer Press, říjen 2005. 264 s. ISBN 80-251-0721-3.

COHN, Mike. *Succeeding with Agile: Software Development Using Scrum.* Addison-Wesley, 2009. 504 s. ISBN 978-0-321-57936-2.

COHN, Mike. *User Stories Applied: For Agile Software Development.* Addison Wesley, 2004. 268 s. ISBN: 978-0321205681.

FIANTA, Roman. *Použití SCRUM pro menší a střední webové projekty.* Brno, 2011. Diplomová práce. Masarykova univerzita, Fakulta informatiky.

HRADECKÁ, Petra. *Metodika pro výběr a nasazení CASE nástrojů.* Praha, 2008. Diplomová práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, Katedra informačních technologií.

KADLEC, Václav. *Agilní programování: Metodiky efektivního vývoje softwaru.* Brno: Computer Press. 2004. 278 s. ISBN 80-251-0342-0.

KANISOVÁ, Hana a Miroslav MÜLLER. *UML srozumitelně.* Brno: Computer Press, 2004. 160 s. ISBN: 80-251-0231-9.

KENDALL, Kenneth E. a Julie E. KENDALL. *Systems Analysis and Design.* 8th edition. Prentice Hall, 2010. 572 s. ISBN 978-01-360-8916-2.

KNIBERG, Henrik. *Scrum and XP from the Trenches: How We Do Scrum.* C4Media Inc., 2007. 140 s. ISBN 978-1-4303-2264-1.

LEFFINGWELL, Dean. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise (Agile Software Development Series).* Addison-Wesley, 2011. 560 s. ISBN: 978-0321635846.

MANNOVÁ, Božena a Karel VOSÁTKA. *Řízení softwarových projektů.* Praha: České vysoké učení technické, 2005. 187 s. ISBN 80-01-03297-3.

MARTINÁSEK, Petr. *Změna metodiky řízení SW projektů.* Ostrava, 2011. Vysoká škola báňská - Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, Katedra informatiky.

PAGE-JONES, Meilir. *Základy objektově orientovaného návrhu v UML.* Grada Publishing. 2001. 367 s. ISBN 80-247-0210-X.

PICHLER, Roman. *Agile Product Management with Scrum: Creating Products that Customers Love.* Addison-Wesley, 2010. 133 s. ISBN: 978-0321605788.

POPPENDIECK, Mary a Tom POPPENDIECK. *Lean Software Development: An Agile Toolkit.* Addison-Wesley, 2003. 240 s. ISBN 978-0321150783 .

RUMBAUGH, James, Ivar JACOBSON a Grady BOOCH. *The Unified Modeling Language Reference Manual.* Essex: Addison-Wesley, 1999. 568 s. ISBN 0-201-30998-X

ŘEPA, Václav a Dušan CHLAPEK. *Materiály ke strukturované analýze.* Praha: Vysoká škola ekonomická. 1997. 138 s. ISBN 80-7079-260-4.

SCHWABER, Ken a Jeff SUTHERLAND. *Software in 30 Days: How Agile Managers Beat the Odds, Delight Their Customers, And Leave Competitors In the Dust.* John Wiley & Sons, 2012. 194 s. ISBN: 978-1118206669.

SCHWABER, Ken a Mike BEEDLE. *Agile software development with Scrum.* Upper Saddle River: Prentice Hall, 2001. 158 s. ISBN 01-306-7634-9.

SCHWABER, Ken. Agile *Project Management with Scrum (Microsoft Professional).* Microsoft Press, 2004. 192 s. ISBN 978-0735619937.

VONDRÁK, Ivo. *Úvod do softwarového inženýrství.* Verze 1.1. Ostrava: Vysoká škola Báňská - Technická univerzita Ostrava. 2002. 73 s.

Elektronické zdroje a ostatní

ADAPTIC. *Co je ROI.* [online]. Praha, Adaptic, ©2013 [cit. 2013-03-13]. Dostupné z: http://www.adaptic.cz/znalosti/slovnicek/roi/

ALDOLF, Filip. *Metodika RUP.* [online]. 03. 02. 2008 [cit. 2013-03-13]. Dostupné z: http://objekty.vse.cz/Objekty/Rup4

ALDOLF, Filip. *Metodika RUP.* [online]. 03. 02. 2008 [cit. 2013-03-13]. Dostupné z: http://objekty.vse.cz/Objekty/Rup2

AMBLER, Scott W. *Agile Modeling (AM) Home Page: Effective Practices for Modeling and Documentation* [online]. Scott W. Amble, ©2012. Dostupné z: http://www.agilemodeling.com

ARRAJ, Valerie. ITIL: *The Basics.* [online]. APM Group Limited, ©2010[cit.2013-03-13]. Dostupné z: http://www.best-management-practice.com/gempdf/ITIL_The_Basics.pdf

BECK, Kent et al. *Manifesto for Agile Software Development.* [online]. Kent Beck et al, ©2001 [cit. 2013-03-13]. Dostupné z: http://www.agilemanifesto.org

BECK, Kent et al. *Principles behind the Agile Manifesto.* [online]. Kent Beck et al, ©2001 [cit. 2013-03-13]. Dostupné z: http://www.agilemanifesto.org/principles.html

BOBEK, Martin. *Jak funguje vývojový tým a programování v Seznamu – Connect.cz.* [online]. Mladá fronta, 24. 1. 2012, ©2013 [cit. 2013-03-30]. Dostupné z: http://connect.zive.cz/clanky/jak-funguje-vyvojovy-tym-a-programovani-v-seznamu/sc-320-a-162045

BUCHALCEVOVÁ, Alena. *Agilní a rigorózní metodiky*. [online]. [cit. 2013-03-13]. Dostupné z: www.nb.vse.cz/~vorisek/FILES/4IT215_materialy_k_predmetu/ MetodikyIT_Buchalcevova.ppt

COHN, Mike a Clinton KEITH. *How To Fail With Agile:Twenty Tips to Help You Avoid Success* [online]. Mountain Goat Software, JUL 1, 2008, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/articles/how-to-fail-with-agile/

COHN, Mike. *Agile Burn Chart - Release Burndown Charts.* [online]. Mountain Goat Software, ©2012. [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/scrum/release-burndown

COHN, Mike. *An Overview of Scrum for Agile Software Development.* [online]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/scrum/overview

COHN, Mike. *Learn About the Scrum Product Backlog.* [online]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/scrum/product-backlog

COHN, Mike. *Learning Scrum - The Product Owner.* [online]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/scrum/product-owner

COHN, Mike. *Managing Risk on Agile Projects with the Risk Burndown Chart.* [online]. Mountain Goat Software, APR 8, 2010, ©2012 [cit. 2013-03-23]. Dostupné z: http://www.mountaingoatsoftware.com/blog/managing-risk-on-agile-projects-with-the-risk-burndown-chart

COHN, Mike. *Planning Poker.* [online]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/topics/planning-poker

COHN, Mike. *Scrum Task Board Training.* [online]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/scrum/task-boards

COHN, Mike. *Scrum Team Training - The Scrum Team.* [online]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/scrum/team

COHN, Mike. *Scrum Training on Sprint Backlog - Sprint Backlog.* [online]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/scrum/sprint-backlog

COHN, Mike. *ScrumMaster: Appointed or Team-Selected?* [online]. Mountain Goat Software, OCT 16, 2006, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/articles/scrummaster/

COHN, Mike. *Sprint Review Meeting.* [online]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/scrum/sprint-review-meeting

COHN, Mike. *The Daily Scrum Meetin.* [online]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/scrum/daily-scrum

COHN, Mike. *User Stories.* [online]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/topics/user-stories

COHN, Mike. *What is Agile Project Management.* [online]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/topics/agile-project-management

COHN, Mike. *What is Scrum Methodology?* [online]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/topics/scrum

DEEMER, Pete, Gabrielle BENEFIELD, Craig LARMAN a BAS VODDE. *The Scrum primer. A Lightweight Guide to the Theory and Practice of Scrum.* [online]. GoodAgile, 2012. Version 2.0 [cit. 2013-03-13]. Dostupné z: http://www.scrumprimer.com/

IBM. *IBM Rational Unified Process* [online]. IBM Corporation, 02-07, ©2007 [cit. 2013-03-13]. Dostupné z: ftp://public.dhe.ibm.com/software/rational/web/datasheets/RUP_DS.pdf

INTERNATIONAL SCRUM INSTITUTE. *Sprint Burndown reports / charts.* [online]. International Scrum Institute, ©2013 [cit. 2013-03-23]. Dostupné z: http://www.scrum-institute.org/Sprint_Burndown_Reports.php

IOS. *ISO/IEC 12207:2008 - Systems and software engineering - Software life cycle processes.* [online]. ISO, 2008 [cit. 2013-03-13]. Dostupné z: http://www.iso.org/iso/catalogue_detail?csnumber=43447

ISO. *ISO/IEC 15504-1:2004-Information technology Process assessment Part 1: Concepts and vocabulary.* [online]. ISO, 2004 [cit. 2013-03-13]. Dostupné z: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=38932

ITBIZ. *Vývoj pomocí SCRUM metodiky využíván i firmami v Česku. ITbiz.cz.* [online]. Argonit, 20. červenec 2011 [cit. 2013-03-30]. Dostupné z: http://www.itbiz.cz/scrum-metodika-agilni-vyvoj. ISSN 1802-1581

KADLEC, Václav. *Rational Unified Process: základní pojmy – Živě.cz* [online]. Mladá fronta, 5. 8. 2003, ©2013 [cit. 2013-03-13]. Dostupné z: http://www.zive.cz/Clanky/Rational-Unified-Process-zakladni-pojmy/sc-3-a-113011/default.aspx

KNESL, Jiří. *Agilní vývoj: Scrum.* [online]. zdroják.cz, 18. 12. 2009 [cit. 2013-03-13]. Dostupné z: http://www.zdrojak.cz/clanky/agilni-vyvoj-scrum/

KNESL, Jiří. *Reference.* [online]. Jiří Knesl, ©2010 [cit. 2013-03-30]. Dostupné z: http://www.knesl.com/articles/view/reference

KRUCHTEN, Philippe. *Architectural Blueprints-The "4+1" View Model of Software Architecture.* [online]. Rational Software Corp., Paper published in IEEE Software 12(6)

November 1995 [cit. 2013-03-13]. Dostupné z: http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

KUCHAŘ, Štěpán. *Řízení kvality IT služeb.* [online]. Vysoká škola báňská - Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, Katedra informatiky [cit. 2013-03-13]. Dostupné z: http://www.cs.vsb.cz/navrat/vyuka/sws/prednasky/pred10a.pdf

MANAGEMENT MANIA. *ITIL (Information Technology Infrastructure Library).* [online]. Management Mania, © 2013 [cit. 2013-03-13]. Dostupné z: https://managementmania.com/cs/information-technology-infrastructure-library.pdf

MANAGEMENT MANIA. *SLA (Service Level Agreement).* [online]. Management Mania, © 2013. [cit. 2013-03-13]. Dostupné z: https://managementmania.com/cs/service-level-agreement

MOUNTAIN GOAT SOFTWARE. *An Introduction to Scrum.* [online prezentace]. Mountain Goat Software, ©2012 [cit. 2013-03-13]. Dostupné z: http://www.mountaingoatsoftware.com/uploads/presentations/English-Redistributable-Intro-Scrum.ppt

MOUNTAIN GOAT SOFTWARE. *Planning Poker*. [online]. Mountain Goat Software, LLC. [cit. 2013-03-13]. Dostupné z: http://www.planningpoker.com/

MOUNTAIN Goat Software. *Planning Poker.* [online]. Mountain Goat Software [cit. 2013-03-13]. Dostupné z: http://www.planningpoker.com/

RICHTA, Karel. SWI041: Modelování a realizace programových systémů. [online]. Univerzita Karlova v Praze, Matematicko-fyzikální fakulta, Katedra softwarového inženýrství. [cit. 2013-03-13]. Dostupné z: www.ksi.mff.cuni.cz/~richta/NSWI041/SWI041-0.pdf

SCRUM.ORG. *What is Scrum?* [online]. Scrum.org, ©2013 [cit. 2013-03-13]. Dostupné z: http://www.scrum.org/Resources/What-is-Scrum

SCHWABER, Ken a Jeff Sutherland. *Průvodce Scrumem: Pravidla hry.* [online]. říjen 2011 [cit. 2013-03-13]. Dostupné z: http://www.scrum.org/Portals/0/Documents/ Scrum%20Guides/ Scrum%20Guide%20-%20CS.pdf#zoom=100

SCHWABER, Ken a Jeff SUTHERLAND. *The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process.* [online]. Jeff Sutherland, 10/14/2007, ©2007 [cit. 2013-03-13]. Dostupné z: http://assets.scrumfoundation.com/downloads/2/scrumpapers.pdf

SCHWABER, Ken. *Scrum As A Framework. Ken Schwaber's Blog: Telling It Like It Is.* [online]. September 8, 2010 [cit. 2013-03-13]. Dostupné z: http://kenschwaber.wordpress.com/2010/09/08/scrum-as-a-framework/

SUCHOR, Jiří. *Analýza a návrh systémů.* [online]. ©2002, 23. 9. 2003 [cit. 2011-04-05]. Dostupné z: http://www.fi.muni.cz/~sochor/PB007/Slajdy/UvodP007.pdf

SUTHERLAND, Jeff. *Scrum Log Jeff Sutherland: Requirements for Product Owner: Common Pitfalls.* [online]. Jeff Sutherland, January 08, 2013, ©2013 [cit. 2013-03-13]. Dostupné z: http://scrum.jeffsutherland.com/2013/01/requirements-for-product-owner-common.html

SUTHERLAND, Jeff. *Scrum Log Jeff Sutherland: Story Points: Why are they better than hours?* [online]. Jeff Sutherland, September 30, 2012, ©2013 [cit. 2013-03-13]. Dostupné z: http://scrum.jeffsutherland.com/2010/04/story-points-why-are-they-better-than.html

SUTHERLAND, Jeff. *Work Less, Get More Done!* [online]. FEBRUARY 13, 2013, ©2012 [cit. 2013-03-13]. Dostupné z: http://scrum.jeffsutherland.com/2013/02/work-less-get-more-done.html

THE STATIONERY OFFICE. *Best Management Practice for portfolio, programme, project risk and service management.* [online]. The Stationery Office [cit. 2013-03-13]. Dostupné z: http://www.best-management-practice.com/

ÚŘAD PRO VEŘEJNÉ INFORMAČNÍ SYSTÉMY. *Standard ISVS 005/02.01 pro náležitosti životního cyklu informačního systému.* [online]. Úřad pro veřejné informační systémy, 11. 12. 2002 [cit. 2013-03-13]. Dostupné z: http://ftp.aspi.cz/aspi/is06-202.pdf

VALLO, Machal a členové agilní komunity Agília. *12 principů Agilního Manifestu.* [online]. Aguarra, ©2001 [cit. 2013-03-13]. Dostupné z: http://www.agilia.cz/wp-content/uploads/2012/05/Agilni-Manifest-12-principu-CZ.pdf

VALLO, Machal a členové agilní komunity Agília. *Manifest pro Agilní vývoj softvéru.* [online]. Aguarra, ©2001 [cit. 2013-03-13]. Dostupné z: http://www.agilia.cz/wp-content/uploads/2012/05/Agilni-Manifest-verze-CZ.pdf

VONDRÁK, Ivo. *Objektově orientované metody.* [online] Ostrava. Vysoká škola báňská - Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 2003 [cit. 2013-03-13]. Dostupné z: http://vondrak.cs.vsb.cz/download/Objektove_orientovane_metody.pdf

VOŘÍŠEK, Jiří. *Jak členit informační služby a navrhovat jejich architekturu.* [online]. Praha, Vysoká škola ekonomická, Katedra informačních technologií, 2008 [cit. 2013-03-13]. Dostupné z: http://si.vse.cz/archive/proceedings/2008/jak-clenit-informaticke-sluzby-a-navrhovat-jejich-architekturu.pdf

Pokud některé z povinných údajů u jednotlivých citací chybí, nepodařilo se mi tyto informace zjistit.

Seznam zkratek

ASD	Adaptive Software Development
AUP	Agile Unified Process
BBC	British Broadcasting Corporation
BPMN	Business Process Model and Notation
CASE	Computer Aided Software Engineering
CD	Compact Disc
COBIIT	Control Objectives for Information and Related Technology
CORBA	Common Object Request Broker Architecture
CMMI	Capability Maturity Model Integration
CMS	Content Management System
CRM	Customer Relationship Management
ČR	Česká republika
DFD	Data Flow Diagram
DSDM	Dynamic Systems Development Method
ERD	Entity Relationship Diagram
ERP	Enterprise Resource Planning
eSCM-SP	eSourcing Capability Model for Service Providers
eTOM	enhanced Telecom Operations Map
EUP	Enterprise Unified Process
FDD	Feature-Driven Development
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
IBM	International Business Machines Corporation
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standards Organization
ISVS	informační systém veřejné správy
IT	informační technologie
ITIL	Infrastructure Library
MOR	Management of Risk
OCL	Object Constraint Language
OID	Object Identifier

OMG	Object Management Group
OO	objektově orientovaný
OOP	objektově orientované programování
OOPSLA'95	Object-Oriented Programming, Systems, Languages & Applications '95
OPEN	Object-Oriented Process, Environment and Notation
PHP	Hypertext Preprocessor
PMBOK	Project Management Body of Knowledge
RAD	Rappid Application Development
ROI	Return On Investments
RUP	Rational Unified Process
SEO	Search Engine Optimization
SEM	Search Engine Marketing
SLA	Service Level Agreement
SPICE	Software Process Improvement and Capability Determination
SW	Software
TDD	Test-Driven Development
UML	Unified Modeling Language
UP	Unified Process
URL	Uniform Resource Location
XP	Extreme Programming

Seznam obrázků

Seznam tabulek

Seznam grafů

Prohlášení o využití výsledků diplomové práce

Prohlašuji, že

- jsem byl seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou práci užít (§ 35 odst. 3);
- souhlasím s tím, že diplomová práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že bibliografické údaje o diplomové práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, diplomovou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

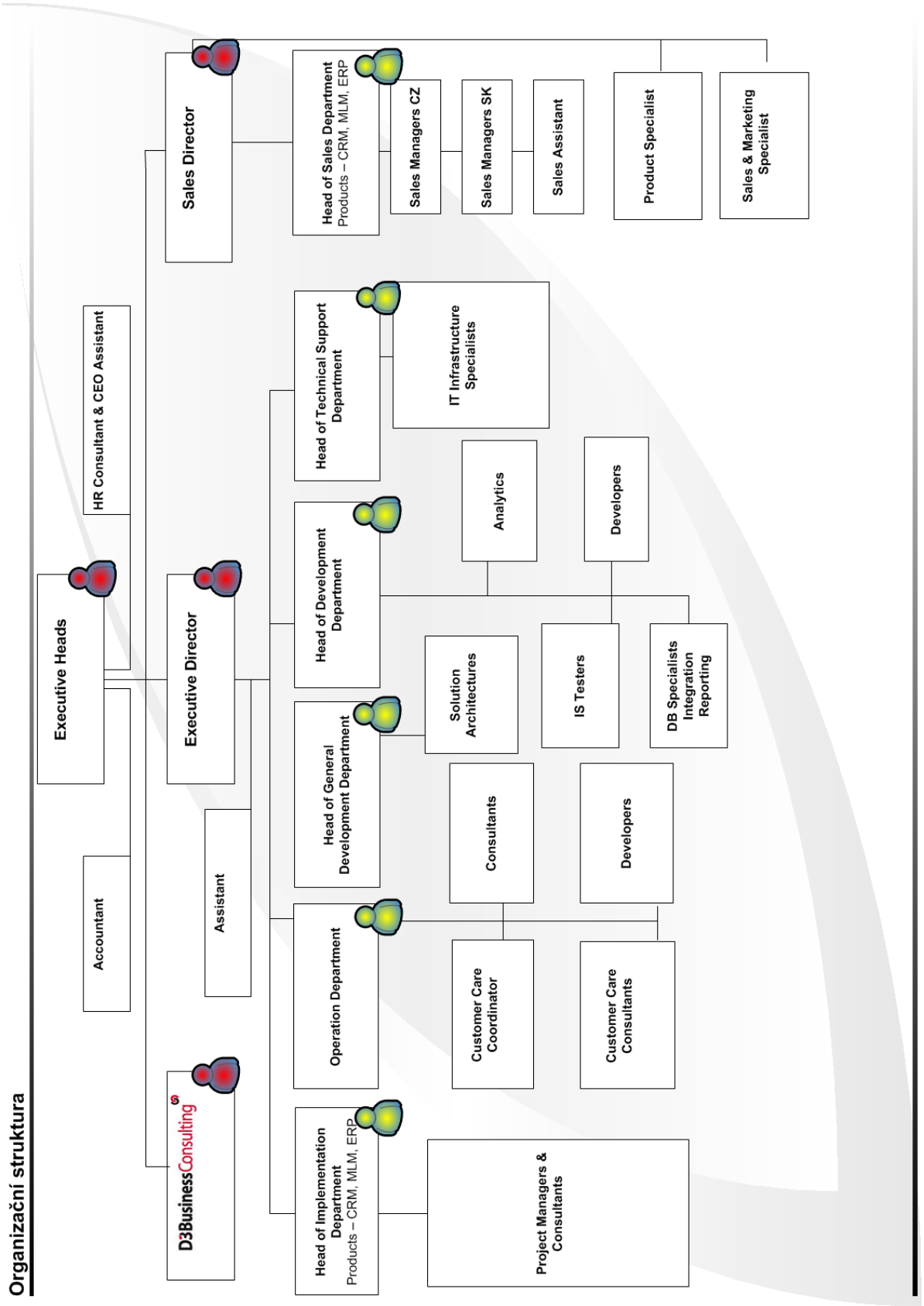
V Ostravě dne 25.4.2013

.....

Bc. Petr Martinásek

Seznam příloh

Příloha A



Obr. A.1 Organizační struktura společnosti D3Soft, poskytnutá firmou